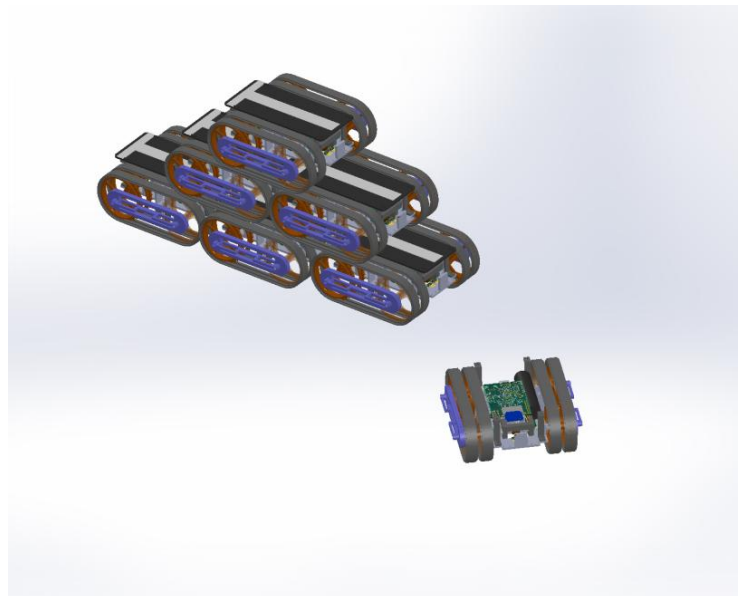


Master Thesis

Microengineering

Towards Self-Assembled Structures with Mobile Climbing Robots



Lucian Cucu

Prof. Radhika Nagpal

Prof. Dario Floreano

Harvard University, August 2014

Contents

1	Introduction	5
1.1	Background and Motivation	5
1.2	Litterature Review	6
1.2.1	Modular or Self-reconfiguring Robots	6
1.2.2	Obstacle climbing robots	7
1.2.3	Pure vertical climbing robots	8
1.2.4	Adhesion and attachment mechanisms	8
1.3	Goals	11
2	Mechanical Solutions for Climbing	12
2.1	Preliminary Reduction of Design Options	12
2.2	Further Reduction of the Design Options	14
3	Robot Model and Geometrical Optimization	16
3.1	Simple Robot Model	16
3.1.1	Case I - Climbing On One Robot	17
3.1.2	Case II - climbing on a structure	21
3.2	Flipper Robot Model	22
4	Mechanical Design	25
4.1	Tracks	25
4.2	Worm drive	25
4.3	Motors	27
4.3.1	Wheels	27
4.3.2	Worm Gear	28
4.4	Chassis	29
4.5	Worm Drive Support	30
4.6	Wheels	30
4.7	Deck	30
5	Embodiment and Climbing Assessment	32
5.1	Climbing Assessment	34
5.1.1	Phase I	34

5.1.2	Phase II & III	36
6	Towards Full Autonomy	39
6.1	Choosing a Structure	39
6.2	Building a 2D Tower	40
6.2.1	Challenges	40
6.2.2	Solutions & Sensors	40
6.2.3	Electronics - Overview	41
6.2.4	Solving the Alignment Problem	42
6.3	Construction Algorithm	44
6.3.1	Implementation	45
6.3.2	Code	50
7	Autonomy Assessment & Discussion	52
7.1	Discussion	53
8	Conclusion & Future Work	55
8.1	Future Work	55
8.2	Acknowledgements	56
A	Code Details	61
	List of Figures	86
	List of Tables	90

Chapter 1

Introduction

1.1 Background and Motivation

Social insects like ants and termites fascinate by their capabilities of constructing massive and complex structures in a decentralized way. These structures are much bigger than the size of one individual (termites mounds easily reach up to 2m high) and are achieved without global coordination. The insects use pheromones and the environment (stigmergy) to communicate locally with each other.

Outside the mound however, structures (and infrastructures) are needed to cope rapidly with new obstacles and situations. Social insects deal with these in an original and pragmatic manner by using their own bodies as immediately-available building blocks.

In the case of an unexpected water flood, fire ants will link together to form a raft. The strength of the linkages between ants and the moderately hydrophobic layer present on their exoskeletons make the raft hold together and float [1]. Japanese honey bees use their bodies to cluster around individual predators (e.g. hornets) thus rising the temperature until the predator is “cooked”. The lethal temperature for bees is around 48-50 °C, whereas for the hornet it is at 44-46 °C [2].

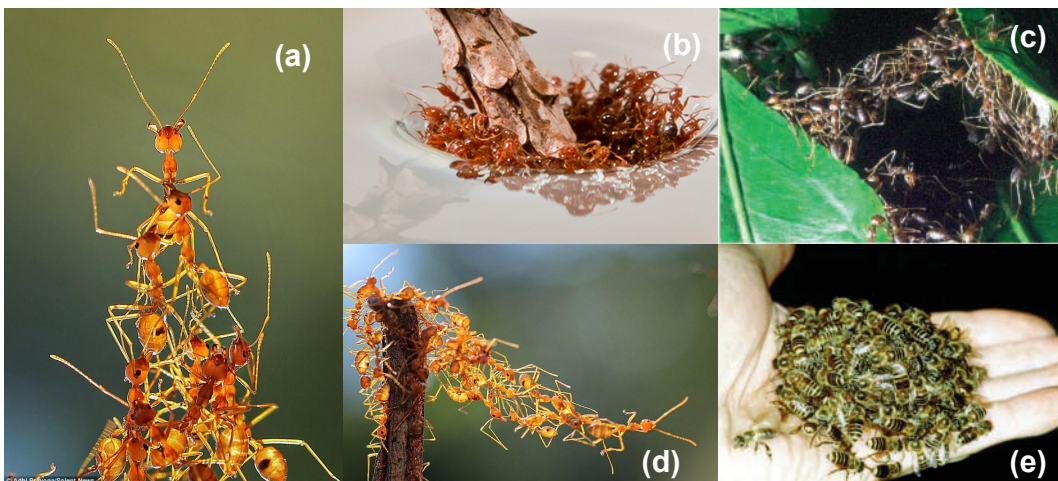


Figure 1.1: (a) weaver ant tower [3]; (b) fire ant raft (pushed into the water with a twig) [1]; (c) army ant bridge [2]; (d) weaver ant chain [3]; (e) Japanese honey bee oven around a hornet [2]

When in difficult situations, army ants can build bridges, ladders, walls and chains. These structures are meant to provide significant logistical advantage when defending, foraging, or emigrating. Towers have been also witnessed when higher objectives had to be reached.

These behaviors provide social insects with a remarkable versatility and adaptivity. In robotics, such behaviour would also be desired in situations where the objective exceeds the physical or computational power of only one individual. Self-assembly through stigmergy could allow a team of robots with limited capabilities to accomplish difficult tasks, with minimal (or no) communication, and without any supervisory control.

Overcoming unpredicted obstacles and sudden changes in the environment represents a major challenge in robotics. Solving such issues would have a significant impact in fields such as exploration or rescue. One solution might be the *ad-hoc* construction of a support structure by other fellow robots, in order to allow others to proceed further. In places where materials are not available, the *body of the robot* itself would be the only accessible *controlled element* in the environment. Hence, it could serve as a reliable building block.

However in order to self-assembly and to build any kind of support structure, the robots have to have specific mechanical features, allowing them to **climb** on each other, without getting stuck. This research will focus on building robots capable of climbing and self-assembling.

1.2 Literature Review

The concept of “climbing” robots is broad one. “Climbing robots” can be capable of moving on vertical surfaces, of transiting from horizontal to vertical ones, or of overcoming obstacles of various shapes and sizes. In the same category can also be included modular, self-reconfigurable robots, which although not capable of climbing individually, are capable of doing it when in more complex formations.

This research is at the crossroad between self-reconfigurable robots and mobile climbing robots. Attachment and adhesion systems play a crucial role for both climbing and driving capabilities, and will be separately investigated. A brief, non-exhaustive overview of these different types is presented in the following section.

1.2.1 Modular or Self-reconfiguring Robots

Some researches have already tackled the problem of climbing from a particular perspective, where emphasis is put less on individual complexity, and more on the possibility of assembling into more complex and versatile structures. Thus, although with very reduced mobility and technically no climbing possibility for an individual module, an assembled system could reach configurations that deal with both issues. A thorough survey of previous systems can be found in [4].

Self-reconfiguring mobile robots can be either *lattice*-like or *chain*-like. Pure lattice architecture is present in works like [5] [6], [7] or [8] to cite only the most recent ones. In the first two ones, the modules rely on an exterior movement in order to reach the desired configuration, the only degree of freedom (DoF) consisting of attaching or detaching. In the last example, the robots are provided with a particular kind of degree of freedom through a momentum wheel. By suddenly braking the wheel the cubes are able

to move on the floor and jump from one lattice position to another. In [9] the robots can slide on top of each other thanks to a rail system. The movement is of course limited to a plane and the modules have to be correctly docked to each other.

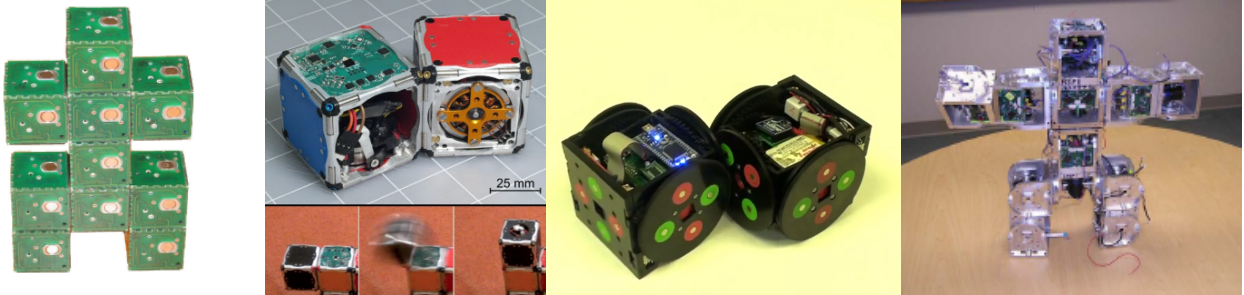


Figure 1.2: *modular robots (from left to right) : (a) lattice-like robot MICHE [5], (b) M Blocks [8], (c) two SMORES modules [10], (d) a complex assembly of SuperBot modules [11]*

As we focus our interest on systems that present some kind of controlled individual mobility in more than 1DoF, the SMORES system ([10], figure 1.2 (c)) is particularly interesting. It is inspired by the SuperBot [11] and the CKbot [12], which can reassemble after destruction, yet it brings the classical modular robot design to a more mobile level, by combing differential drive with a docking mechanism. Additional degrees of Freedom for panning and tilting, as well as a symmetric design and an underactuated docking mechanism make it an interesting forward step in the field.

1.2.2 Obstacle climbing robots

On the other extreme, some robots achieve remarkably robust obstacle climbing without compromising their mobility. The Shrimp uses parallel articulations and rocker-boogies in order to passively overcome obstacles up to twice its wheel size [13].

Other robots are designed for more controlled obstacle climbing such as stairs. [14] uses tanks treads and a third intermediate wheel which helps the robot change its center of mass. The wheel is also in the same plane with the tracks and can be moved with the help of a lever, thus providing some kind of reconfigurability to the robot. In order to keep the third wheel under one track, a spring-loaded prismatic joint is used.

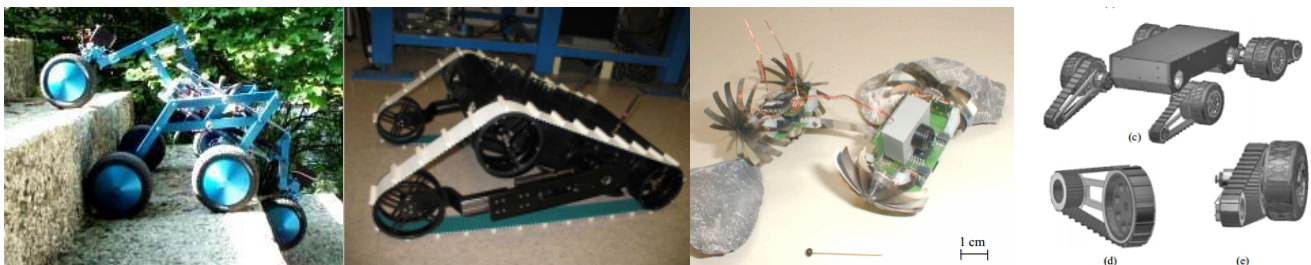


Figure 1.3: *climbing robots for rough terrain (from left to right): (a) SHRIMP, (b) reconfigurable tank-tread robot, (c) LaMalice, (d) Mobit*

An interesting compromise between modularity and climbing capabilities has been proposed in [15], where 1DoF tank-treaded modules can use friction to change configuration. A similar concept is presented

in [16] and [17] where tracked modules hook together to be able to steer and overcome obstacles.

Sometimes more complex mechanics are used in tank-treaded robots to be able to change locomotion mode. In [18] the system uses a pivot to change drive configuration from track-only to wheels and tracks. Consequently, it is able to tackle various obstacles, reaching heights up to twice its wheel diameter.

LaMalice presents a design combining whegs and a flexible spine in a light-weight (30g), energy-efficient (50mW) and compact form (11cm long, 6 cm wide, 4 cm high), while being able to overcome obstacles of its own height [19]. As the spine bends, it allows for a better gripping and a “push-pull” effect is achieved, which facilitates climbing. As the whegs are flexible, a movement of the center of mass occurs, which increases the smoothness and decreases the power consumption when surmounting an obstacle . The robot is also designed such that it can be coupled to other modules.

MOBIT is an interesting example where legs, tracks and wheels are combined to achieve maximum versatility and mobility [20]. It uses 4 wheels adjacent to 4 tracks, each one with an actuated flipper, allowing a wide span of different locomotion possibilities.

1.2.3 Pure vertical climbing robots

Once fixed on the respective surface, these robots will be able to travel on the entire surface, in order to perform maintenance tasks such as cleaning, gritting or surface inspection. The most used adhesion mechanisms include active suction cups [22], [23] using dry or wet[24] adhesion, and permanent magnets [25]. The reliability of such systems has been proven, allowing the entrance on the market of such systems. Patented industrial designs as the one from International Climbing Machines (ICM) achieve impressive weight/payload ratios and tackle a wide range of surfaces and obstacles [21].

However these adhesion systems require considerable energy and space, severely limiting their miniaturization perspectives and autonomy. Moreover, for most systems the types of usable surfaces is also limited and obstacles (small edges, contours protrusions) present serious, often insurmountable challenges.

1.2.4 Adhesion and attachment mechanisms

Climbing cannot be achieved without a reliable adhesion system. A detailed review is now given of different adhesion and attachment mechanisms. The goal is to gain insight in some of the existing solutions, in order to eventually take inspiration when designing the climbing robot.



Figure 1.4: *ICM wind turbine inspection robot using vacuum to climb on brick, metal, or concrete [21]*

Pneumatic adhesion using active suction cups has the advantage of coping with various kinds of surfaces while achieving adhesion control through pressure measurement. Problems however occur on porous or dirty surfaces, and the vacuum pumps are bulky and energy voracious. Some promising results have been achieved using the Bernoulli effect, where the force/weight ratio outruns nearly all other known methods [26]. Using a flow of 51 l/min at 5 bars through a 6mm nozzle, an attraction force up to 12N is achieved on a 230g robot. However, constant flow and distance to the (flat) surface have to be maintained.

Magnetic adhesion has been implemented in robotic systems with permanent magnets [25], switchable magnets [5], or electropermanent magnets [6]. Switchable magnets have been used in reconfigurable robotic systems. The one implemented in Miche[5] uses 2 permanent magnets with a switch connecting them, capable of mechanically toggling the magnetic field. The Robot Pebble system uses an electropermanent magnetic system, composed of both a magnetically hard (eg. Nd-Fe-B) and a semi-hard (eg. Alnico) material [6]. The semi-hard magnetic material can be polarized with a help of a coil, thus creating a magnetic field that either reinforces or cancels the strength of the neighboring permanent magnet.

Permanent magnets have been used either on the body [27], wheels [25], or tracks of robots [28].

In modular robotics, permanent magnets are often used for alignment [12]. Despite the high force density available and the energy-efficiency of such systems, detachment remains an issue, specially when considered from the perspective of self-assembling robotic systems.

Mechanical gripping with robotic hands has been widely explored on specific structures such as pipes [29], trusses or poles [30],[31],[32],[33] or even trees [34]. These systems are often composed of complex mechanics and work only on regular and controlled surfaces.

Furthermore, efforts have been done to extend the use of gripper robots to a wider choice of terrain. In [35] a bipedal robot with two grippers as legs uses the same kinematics for climbing and walking. The grippers can grasp or roll, and are used as claws or whigs respectively, depending on the roughness of the terrain. The control of such a device remains complex and in this case it is done through remote control.[34].

A gripper together with a telescopic systems and cables can make a robot like the Treebot climb on a wide variety of trees (e.g. bamboos) , with different diameters while capable of steering and carrying a payload up to three time its own weight

Electroadhesion has been used to keep a 220g on the wall while traveling in one direction [36]. Electroadhesion has the advantage of generating high clamping forces ($0.2-1.4 \text{ N/cm}^2$ depending on the substrate) while requiring relatively low power in the order of $20 \mu\text{W/N}$ of held weight). It is also robust to dust and works both on rough and smooth, as well as conducting and insulating surfaces. However, it requires compliant electronics and

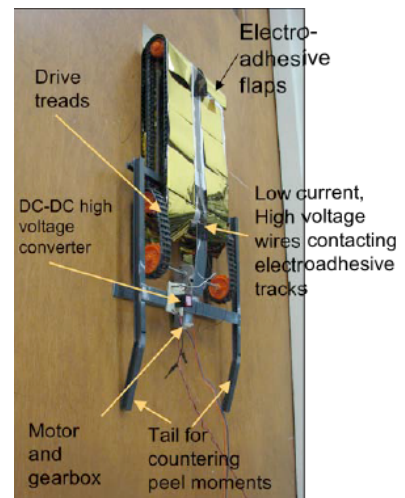


Figure 1.5: *electroadhesive pads implemented on a 1DoF robot*

well-designed mechanics to guarantee close contact of the electrodes to the surface. Here, a belt lays the electroadhesive parts in front of the robot as it advances.

Passive suction cups have proven to work if the mechanical design is suitable. In [37] a 300g robot uses passive suction cups attached on a tread for adhesion and 1DOF locomotion 1.6. As the tracks moves, the next suction cup is laid, while the one at the end of the robot is peeled off. A string interconnects the cups, maintaining a the cups in a defined position and while in the same time enhancing detachment (peeling off).

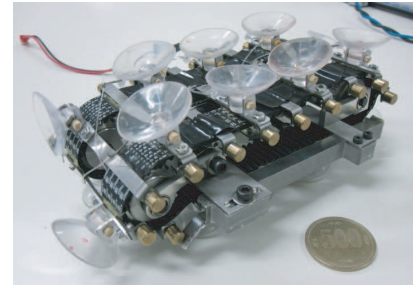


Figure 1.6: *robot with passive suction cups*

Different types of **Bio-inspired** adhesion systems have also been successfully implemented. These types are either mechanical (thin claws) or are based on physicochemical principles (van der Waals forces). In [38] a spider-like robot uses 6 10-toed legs to climb concrete, brick and masonry walls. Each toe is equipped with $200\ \mu\text{m}$ thick shafts ($25\ \mu\text{m}$ at the tip), and designed in a very compliant way to guarantee adhesion to the wall's asperities. Using similar principle, a more advanced version called RiSe, has been built [39].

In [40] $380\text{-}\mu\text{m}$ thick lamellae are etched out of DPS, and installed on the paws of a gecko-like robot. Their specific orientation and size make that van der Waals forces alone suffice to hold the whole 370 g robot body at vertical on smooth surfaces (e.g. glass). The method is however vulnerable to dust and dirt and requires regular cleaning.



Figure 1.7: *different ways of gripping and attaching : (a) bipedal robot with 2 grippers [35], (b) RiSe robot [39], (c) Treebot [34], (e) GeckoBot [40]*

1.3 Goals

Although a considerable number of researches have focused on collective robotic behaviour, and climbing robots independently, none has been conducted thus far around collective behaviors implying mobile robots individually climbing on each other.

Motivated by insect behaviour, we explore the possibility of self-assembly and structure formation with robot bodies as building blocks.

In order for this kind of system to be scalable and to output relevant results, it is crucial that by a thorough optimization effort the number of robots required to achieve a structure of a given characteristic is kept to a strict minimum. The structures that will be considered will be towers or ramps, so the relevant metric here will be *height*.

Hence, the main goals of this project can be enumerated as follows:

1. design, build and assess a robot that is capable of robustly climbing on other *identical* ones
2. optimize its design such that it minimizes the number of robots required for a given height
3. demonstrate a fully autonomous structure formation
4. consider scalability for all design choices
5. reduce as much as possible the dependency on the environment and avoid docking systems

In the first objective, the term *robust* implies that climbing should not only be reliable and repeatable, but that it has also to be so from any given (horizontal) position.

Scalability has to be constantly considered, as it represents the core and essence of the collective robotics. By scalability it is implied that all possible issues that can occur when scaling up the system have to be considered, as much from the mechanical, as from the electrical, sensing and algorithmic perspectives. From a hardware point of view, simple but robust mechanics are targeted, and from a software perspective, algorithms should easily scale up, and be independent of the desired shape size or the number of robots involved, while in the same time tolerant to individual failures.

Finally, the robot should be designed in such a way that self-assembly and structure building can occur in different environments. Ideally, obstacles should be *integrated* in the structure, thus being more of a support than a hindrance. Because of their mechanical designs, little margin for error or imprecision is allowed in modular robotics. The environment has to be controlled, and random obstacles are a serious challenge. Therefore, in order to distance ourselves from modular robotics, docking and locking mechanisms will be avoided.

Chapter 2

Mechanical Solutions for Climbing

2.1 Preliminary Reduction of Design Options

As the literature review points it out, many different climbing methods exist. It is now essential to look into some of them and qualitatively assess which ones would fulfill the criteria required to climb on other identical looking fellow robots.

Before doing so however, the research has to be narrowed down and only the most relevant cases will be discussed. Figure 2.1 shows which solutions are *a priori* excluded and gives an insight on the different climbing mechanics available.

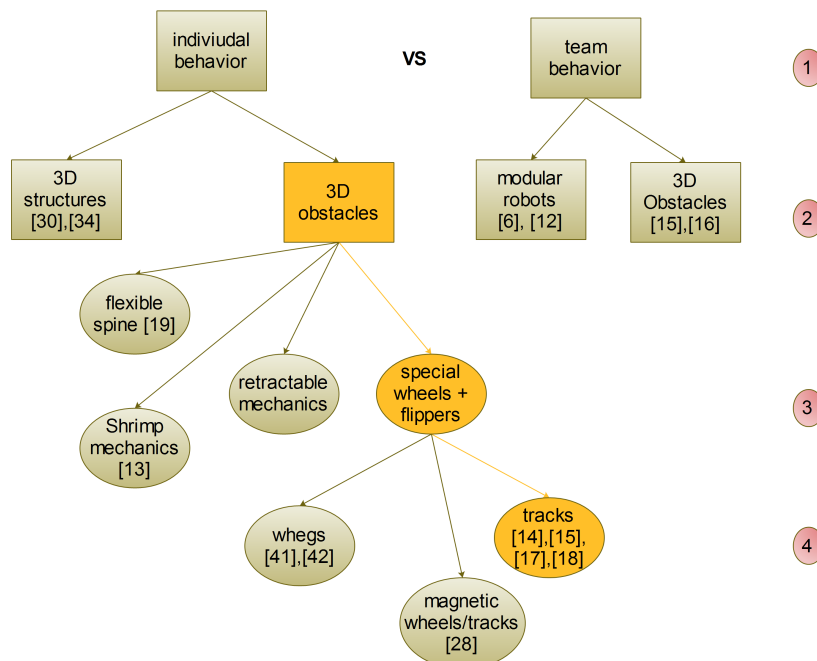


Figure 2.1: overview of some of the climbing categories (rectangle) and some of the common-used mechanics for climbing (circles); the chosen path is highlighted in orange; literature references are added between brackets

Referring to the number bullets on the right side of 2.1, the following steps are justified as follows.

Step 1. It is to be decided if the robot will be able to climb obstacles on its own or only with help of others, as in the example of [15] or [16], or as in the case of modular robots ([12], [10]).

Because one of the goals is to minimize the number of robots implied in any task, it is decided that robots should be able to climb individually, with no kind of prior collaboration. As mentioned in the goals, modular robots are avoided by default.

Step 2 On this level it is decided what kind of climbing the robot should perform. Robots like [30] and [34] can individually climb on trusses and trees respectively and are capable of both vertical and horizontal locomotion if the structure is adequate. As the goal of this project is to create a structure out of robot bodies, and with minimum docking, it is most likely that the end structure will be relatively unpredictable and present irregularities, making these kind of climbing systems inappropriate.

The robots which deal with 3D obstacles matching their own height size interest us the most.

Step 3 On this level, designs for rough terrain locomotion are studied. A mechanical system as the one used in the Shrimp robot [13] could be appealing by its versatility and compliance. By its intrinsic mechanical design, no planning or sensing is needed before tackling an obstacle. Nevertheless, the mechanics are rather complex, which make it difficult to scale to a larger number, and it is expected that because of its legs, the robot would perform poorly if it had to climb an identical fellow.

Another idea would be to use a retractable mechanism. The robot would deploy a small platform (or leg) and push itself on the obstacle. The leg would be retracted afterwards. Although some sensing would be required, climbing on high obstacles would be straightforward and a multitude of this kind of robots would easily pile up without getting stuck or entangled. Yet space and actuation consideration have to be carefully thought over. No such examples have been found in literature.

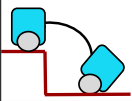
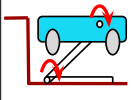
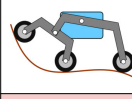
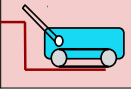
Model	inter-compatibility	climbing	mechanical complexity	sensing and planning	aspect ratio
	1	3	5	5	1
	4	5	1	3	4
	1	4	0	5	3
	5	3	4	3	5

Figure 2.2: *first selection among mechanical designs; the marks are given from 0 to 5, where 5 solves the topic in the most satisfying way; the chosen design is highlighted in red*

Using a flexible spine, as implemented in LaMalice [19] would have the advantage of being mechanically considerably easier to implement, but the robots would have problems piling up, as not enough flat surface would be available for stable support. Moreover, the robot would have to be long, which goes against the goal of minimizing the aspect ratio.

Finally, flippers have the advantage of being mechanically relatively simple to implement, and the robot can be designed such that a good aspect ratio is ensured and enough surface is available for stably supporting the others on top. The robots would be easily compatible in greater number, as the flipper would stay retracted, reducing the risks of entanglement. Independent from the flipper, a variety of wheels can be used (whogs, tracks (magnetic or not), normal wheels). Additionally, flippers could help the robot return to the normal position, which is crucial in an environment where falling and overturning will occur often.

The chart from 2.2 resumes all these considerations. Given the above-mentioned advantages, a robot with flippers is selected.

Step 4 An appropriate locomotion system is now discussed.

Half circle whogs result in impressive results when in a large number and individually controlled, as is the case in the Rhex robot [41]. Whogs have already been used in a smaller number and synchronously actuated ([42]), yet gripping and alignment constitute serious issues to climbing. In the current work, whogs have to be discarded mostly because of their non-convex form, which would make efficient robot piling difficult.

If ferromagnetic surfaces are available, tracks with magnets would provide with remarkable climbing and horizontal to vertical transitions [28]. However, in the context of one robot climbing on another, the nature of magnets (strength of the field proportional to $\frac{1}{d^2}$ make such a solution difficult to engineer. The system should attach enough to climb but should also detach easily enough when a robot would want to get off a neighbour. This is even more complicated when dealing with multiple robots in uncontrolled positions and orientations.

It is decided to opt for a design implementing **simple rubber tracks**. These can be easily found on the market, are low-cost, and require only a straightforward mechanical design. Furthermore, identical tracks can be mounted on the flippers without significant mechanical redesign, thus further enhancing climbing

2.2 Further Reduction of the Design Options

Several flipper-based robot designs are now explored into more detail. The key aspects to consider are the following:

- climbing capability
- getting stuck - specially on corners and edges

- mechanical complexity
- sensors integration - how many sensors, and how much planning is required

Figure 2.3 roughly illustrates the considered possibilities.

(a) is the basic tank-treaded robot. Practical manipulation shows that it gets often stuck on the corners and edges of the other robot, with the tracks losing contact to the ground

To remedy this, solutions (b) and (c) are considered. In (b) the tracks are widened such as to cover all of the robot surface. While dealing of the problem of getting stuck, integrating sensors becomes a serious challenge. In (c) a third track is integrated, shorter than the other ones and not touching the ground when the robot is laid horizontally on flat surface. This track would be useful to help the robot tackle with corner and edge problems.

(d) implements simple flippers, but tracked flippers like in (e) can be mounted with little increase in mechanical complexity and offer the advantage enhanced climbing.

(f) requires more actuators, and the increase in design complexity is not really worth the increase in climbing capability, as 1 of the additional track pairs has a strict length constraint (otherwise it would not fold back). When the aspect ratio becomes high (shorter robot, taller wheels), the additional length provided by this intermediate pair of tracks becomes almost insignificant.

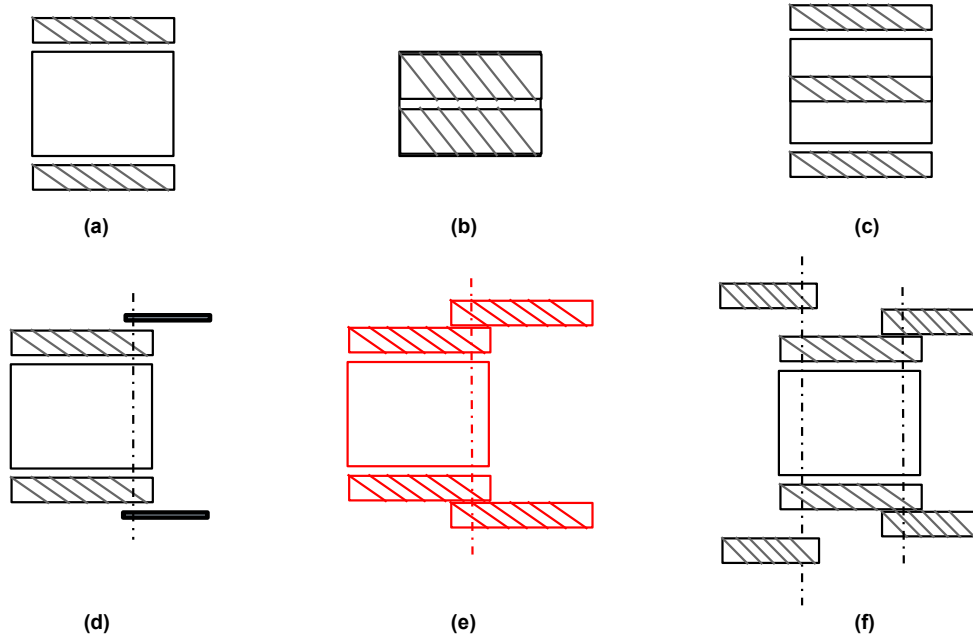


Figure 2.3: *flipper-based design possibilities; the chosen solution is highlighted in red*

In the next chapter a model is proposed for the robot and a theoretical analysis is conducted in order to assess the importance of adding an extra feature (flippers) to a normal tank-treaded robot.

Chapter 3

Robot Model and Geometrical Optimization

As one of the goals of this work is to minimize the number of robots necessary to achieve a structure of a given height, the size and aspect ratio will play an essential role. Hence, a preliminary theoretical analysis and modeling is required in order to choose the optimal solution. A model of the robot would also provide with a metric and a reference for subsequent assessments and comparisons with other designs.

The model developed below is inspired from [16] and is meant to analyse the climbing capabilities of a simplified tracked robot based on its aspect ratio.

The final goal is to **maximize the aspect ratio** by defining the possible design space and finding its optimum. Intuitively, the position of the CoM and the available friction will both help determine the choice of the aspect ratio. The following developments will show how the position of the CoM dictates the geometrical limit of the aspect ratio, and how friction evolves for different ratios.

Once the analysis is over, the embodiment of the robot will be done such as to be as close as possible to the optimal size (aspect ratio, position of CoM) and with the adequate materials (to ensure friction).

3.1 Simple Robot Model

The 2D model in figure 3.1 is considered. The body of the robot does not exceed the diameter of the wheels and symmetry on both sagittal and horizontal plane is assumed. On the frontal plane, only the displacement of the center of mass prevents symmetry.

Let l be the length between the front and rear wheel, r the radius of the wheel, θ the angle of the robot with respect to the ground, d the offset between the geometrical center of the robot and its center of mass (CoM), and μ the friction coefficient. F_1 and F_2 are the normal support forces. It is assumed that no motor torque is used for bending the tread, and that the rolling coefficient of friction is negligible. The obstacle is considered to be a vertical wall of height H , with the same friction coefficient μ as the ground.

The following additional assumptions are made: **(A1)** the robot disposes of unlimited torque; **(A2)** climbing can be decomposed in a set of infinitesimal static steps, where no acceleration or angular mo-

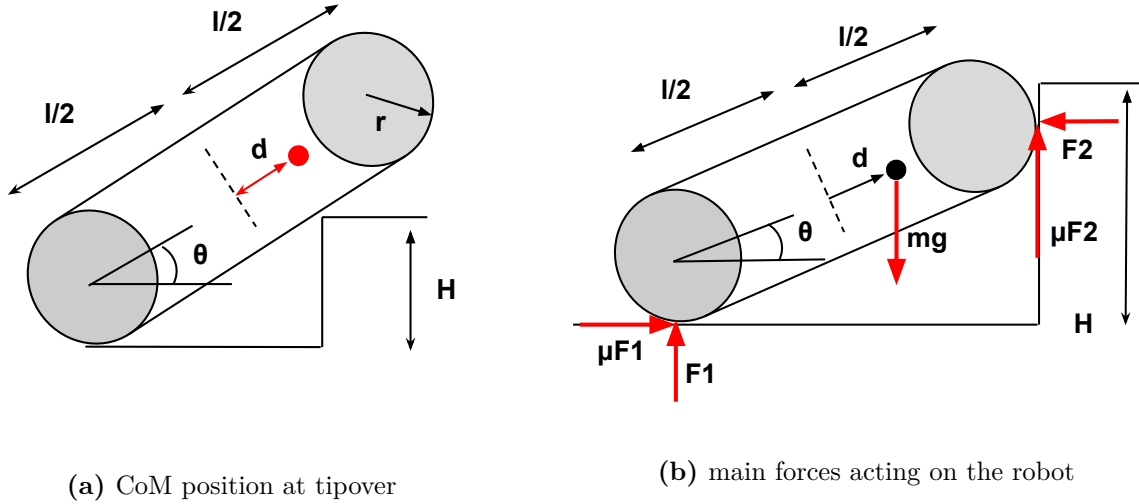


Figure 3.1: *simplified robot model; two conditions have to be respected in order to climb: the CoM has to reach the edge of the obstacle and the robot has to be in static equilibrium at all time*

mentum occurs; **(A3)** in each infinitesimal step only static friction occurs

Therefore, in order for climbing to be successful, the following conditions have to be fulfilled:

1. *tipover*: the CoM of the robot has to go past the corner of the obstacle (figure 3.1a)
2. *equilibrium*: for every infinitesimal angle θ_i static equilibrium has to be met (figure 3.1b)

A consequence of the first condition is that the position of CoM (d) will dictate the geometrical limit of the aspect ratio. It represents the necessary condition for climbing. It results from the second condition that the aspect ratio will also be limited by the friction μ . In the following chapters, a thorough mathematical analysis of both of these aspects is done, and an optimal aspect ratio curve is derived, which will guide the design of the robot. The final result will yield the minimum friction needed for a robot with a given position of the CoM to climb an obstacle of its own height.

Two cases have to be differentiated: in the first one, the robot has only to climb on one identical robot, and in the second case, it has to climb on a structure that contains several identical robots. In both cases, the analysis is entirely in 2D.

3.1.1 Case I - Climbing On One Robot

In a first phase, the geometrical limit to the aspect ratio relative to the position of the CoM is derived. From the tipover condition and figure 3.1a the height can be expressed as:

$$H = r + (l/2 + d) \cdot \sin(\theta) - \frac{r}{\cos(\theta)} \quad (3.1)$$

In order to solve the problem mathematically, the following constraints are used:

$$H \geq 0 \quad (3.2)$$

$$L \geq 2 \cdot r \quad (3.3)$$

$$|d| \leq \frac{L}{2} \quad (3.4)$$

where H is the height of the obstacle, l the length of the robot, r the radius of the wheel and thus half of its height, d the offset to the center of mass with respect to the center of the robot, as shown in figure 3.1b

Condition (3.3) implies that minimum length of the robot is given by the diameter of its wheels, and condition (3.4) implies that the CoM cannot be farther than the axis of the wheels.

Equation (3.1) yields, for an obstacle of the same size as the robot, where $H = 2r$:

$$\frac{\sin(\theta) \cdot \cos(\theta)}{\cos(\theta) + 1} = \frac{r}{l/2 + d} = \frac{r}{l/2(1 + p)} \quad (3.5)$$

where $d = l/2 \cdot p$, with p the position of CoM in percentage of the half-length

With the aspect ratio defined as $k = \frac{r}{l/2} = \frac{\text{robot height}}{\text{robot length}}$, this can be rewritten as:

$$k = \frac{\sin(\theta) \cdot \cos(\theta)}{\cos(\theta) + 1} \cdot (1 + p) \quad (3.6)$$

Figure 3.2a illustrates equation (3.6) for a fixed position of the CoM. The maximum of this function (θ_{lim}) is in the same time its limit, as the function does not have any physical sense past it. Past this angle, the robot is physically not capable of climbing anymore (i.e. to bring its center of mass over the obstacle). In other words, for this given position of the CoM, a even higher aspect ratio would imply reaching an even higher angle θ , which is not possible.

By replacing θ_{lim} in (3.6), the aspect ratio limit k_{lim} relative to p can be derived :

$$k_{lim} = \frac{\sin(\theta_{lim}) \cdot \cos(\theta_{lim})}{\cos(\theta_{lim}) + 1} \cdot (1 + p) \quad (3.7)$$

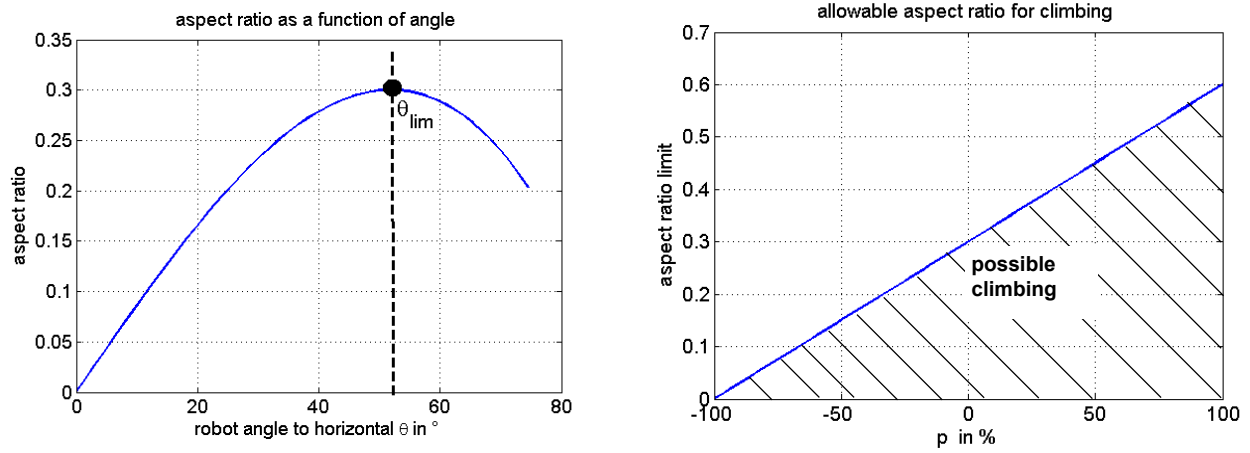
The relationship between k_{lim} and p thus a linear one. Figure 3.2b illustrates this.

In a second phase, the needed friction to climb relative to the aspect ratio is derived.

For the equilibrium condition to be met, a sufficient amount of friction has to be present. By summing the forces and torques represented in 3.1b to zero, μ can be derived :

$$\mu F_2 + F_1 = mg \quad (3.8)$$

$$\mu F_1 = F_2 \quad (3.9)$$



(a) relationship between angle and aspect ratio for a given position of the CoM (here $d = 0$); the function has a physical sense only until θ_{lim}

(b) the limit of the aspect ratio relative to the position of the CoM; beyond the blue line, the necessary *tipover* condition is no longer respected

Figure 3.2

$$\mu F_1 \cdot \left(\left(\frac{L}{2} + d \right) \sin(\theta) + r \right) - F_1 \cdot \cos(\theta) \left(\frac{L}{2} + d \right) + \quad (3.10)$$

$$F_2 \cdot \left(\frac{L}{2} - d \right) \sin(\theta) + \mu F_2 \cdot \left(\left(\frac{L}{2} - d \right) \cos(\theta) + r \right) = 0 \quad (3.11)$$

From combining (3.8) and (3.9):

$$F_1 = \frac{mg}{1 + \mu^2} \quad (3.12)$$

$$F_2 = \frac{\mu \cdot mg}{1 + \mu^2} \quad (3.13)$$

Which yields, when replacing into (3.11) and simplifying:

$$\mu^2((1 - p)\cos(\theta) + k) + \mu(2\sin(\theta) + k) - (1 + p)\cos(\theta) = 0 \quad (3.14)$$

Finally, as this a simple 2nd degree function, for a chosen aspect ratio k , μ can be extracted using the well-known formula:

$$\mu = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (3.15)$$

with only the positive value of μ being considered

The results of (3.14) for a fixed position of the CoM are illustrated in 3.3.

All friction values above the drawn line satisfy the equilibrium condition. The line stops at a certain value which corresponds to k_{lim} computed earlier. The design space is the result of the combination of

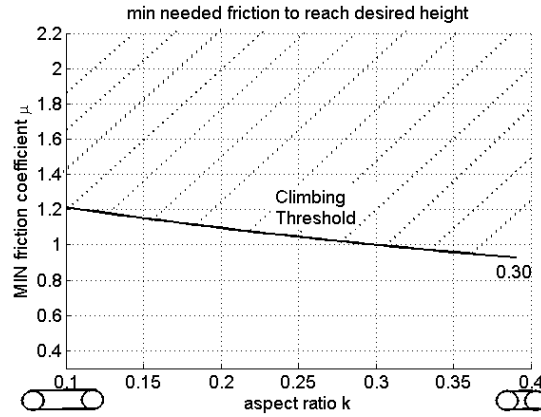
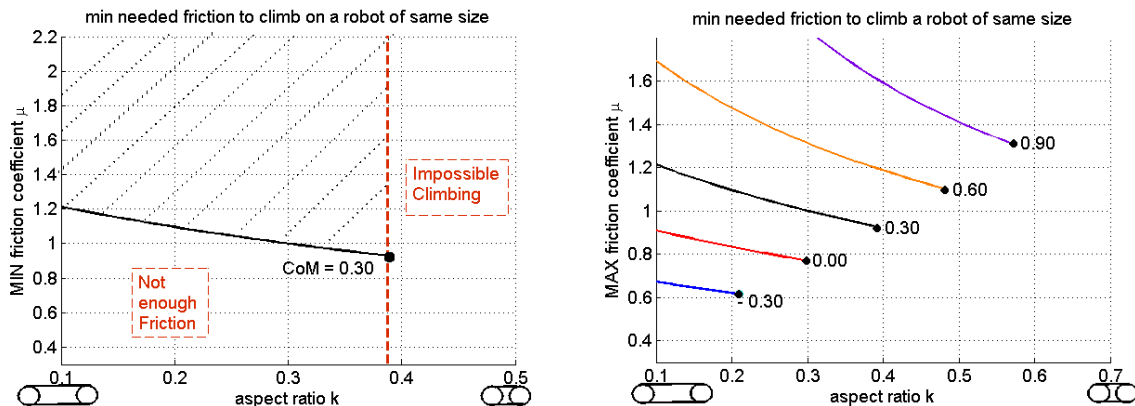


Figure 3.3: illustration of function (3.14) : necessary friction to climb with a given aspect ratio; all points in shaded region are valid

the geometrical and friction conditions and it is drawn in figure 3.4a

It can be noticed that the point where the friction is lowest corresponds also to the biggest aspect ratio, and represents the optimal geometry given a constant offset of the CoM. This is due to the monotonically decreasing function $\mu(k)$ (equation (3.15)) and intuitively, to the fact that larger friction forces are necessary to hold a longer object inclined against a wall. This is doubly convenient, however each line stops at a specific k_{limit} , after which the geometry of the robot does not allow tipover any more. Figures 3.4b shows how this optimal design point varies with the position of the CoM and 3.5b illustrates the optimal design line.

The more the CoM is located at the back, the lower is the friction required to climb, hence the “easier” is the climbing. However, the maximum reachable aspect ratio k_{limit} is smaller. This ratio is only given by the geometric constraints on the tipover condition. The more the CoM is located at the front, the larger can the aspect ratio be, but climbing requires a higher friction.



(a) illustration of a minimum friction line for a fixed offset of the CoM of 30% of L ; all points in shaded region are valid for climbing

(b) illustration of several minimum friction lines for different offsets of the CoM, from -30 to 90 of L %

Figure 3.4: variation of the minimum coefficient of friction necessary to climb with respect to the aspect ratio $k = \frac{height}{length}$

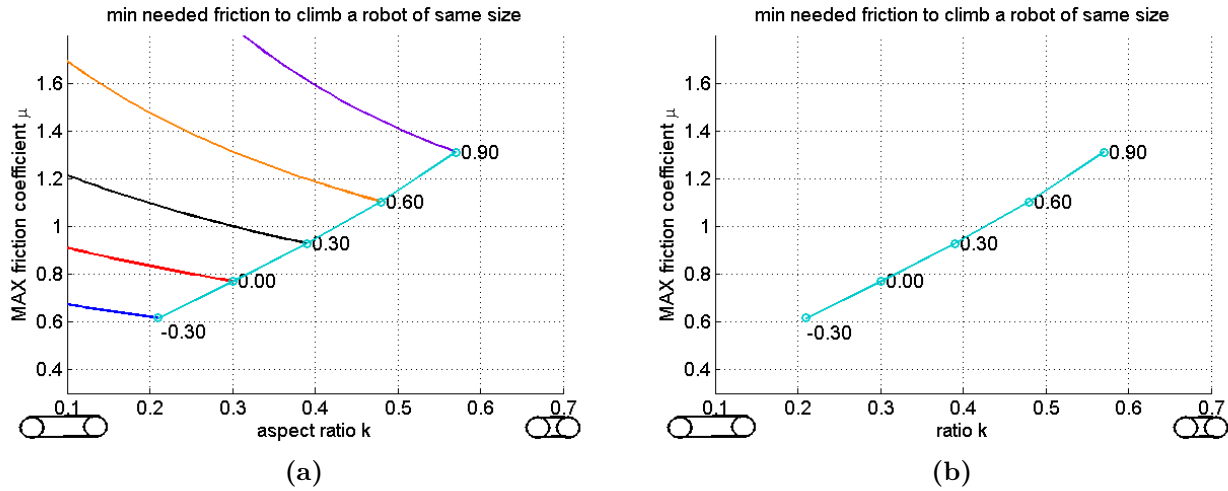


Figure 3.5: optimal design line: for each offset of the CoM the last point of the minimum friction line is taken (left) and represented on a separate curve (right)

3.1.2 Case II - climbing on a structure

Ultimately all robots should be able to assemble into a structure. Being able to climb on one robot is not sufficient to ensure successful climbing on a structure, as this depends on the position that each one occupies (e.g. a robot could not climb on two other robots stacked exactly on top of each other). Simplifying the problem to a 2D pyramid, the necessary conditions on the overall configuration of the structure can be derived.

Two possible climbing scenarios can occur:

- the robots form a *long staircase* structure, where the length of each step is the equal to the entire length of the robot
- the robots form a more *short staircase structure*, where each step length is nor long enough for the robot to return back to the horizontal position, but neither short enough to make climbing impossible

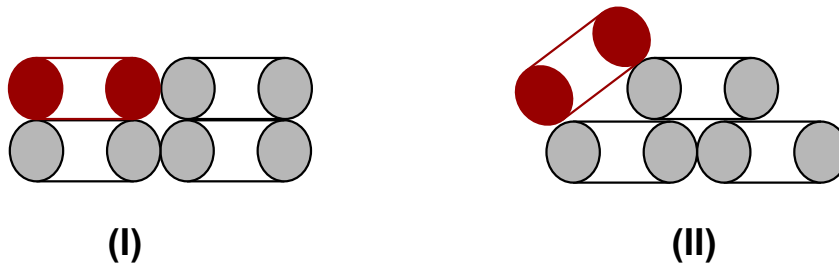


Figure 3.6: two possible structure configurations: either the robot can return to horizontal position as an intermediate step of the ascension, or it continues the ascension with the same or smaller orientation angle; the climbing robot is represented in red

These two scenarios are illustrated in figure 3.6. In the first case, the robot is able to return to the initial horizontal position before tackling the next climb. No additional conditions are required for suc-

cessful climbing.

In the second case, the robot continues on climbing with an equal (or smaller angle), without returning to the horizontal position. This requires new constraints on the length and friction.

For simplicity, the other robots are assumed here to have right-angled edges as in a staircase. At the moment when the CoM has reached the tip of the first stair, the nose of the robot should already be above the second tip as shown in figure 3.7.

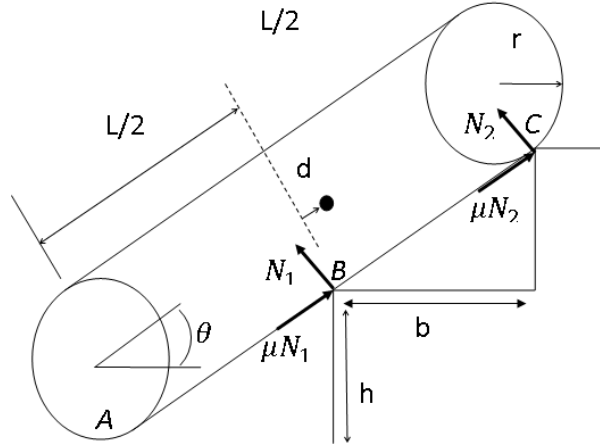


Figure 3.7: *simplification of climbing on several layers of robots; as in a standard inclined plane problem, the friction coefficient has to be larger than the tangent of the inclination angle for the robot to hold still*

At this moment, no forces exist anymore on point A and the configuration of the support forces changes. While the one on B stays the same, the support force on C will be perpendicular to the robot track. In order for the static equilibrium to be held in this new configuration:

$$\mu(N_1 + N_2) = mg \cdot \sin(\theta)$$

with N_1, N_2 the normal support forces as shown in 3.7

As in a standard “ramp” problem, $N_1 = N_2 = \cos(\theta)$ which yield in :

$$\mu \geq \tan(\theta) \tag{3.16}$$

which is the necessary condition for climbing on several layers of robots. This additional condition has an impact on the allowable aspect ratio, as shown in figure 3.8. It is a result of adding this new condition on the design space from illustrated in 3.5a.

It can be observed that this additional condition has significant impact only for the cases where the CoM is at the back of the robot, as in these cases the angle to be attained is higher.

3.2 Flipper Robot Model

The previous chapter has outlined the main performances that can be achieved with the most basic of robots. It has been demonstrated that by putting the CoM at the back of the robot, less friction is

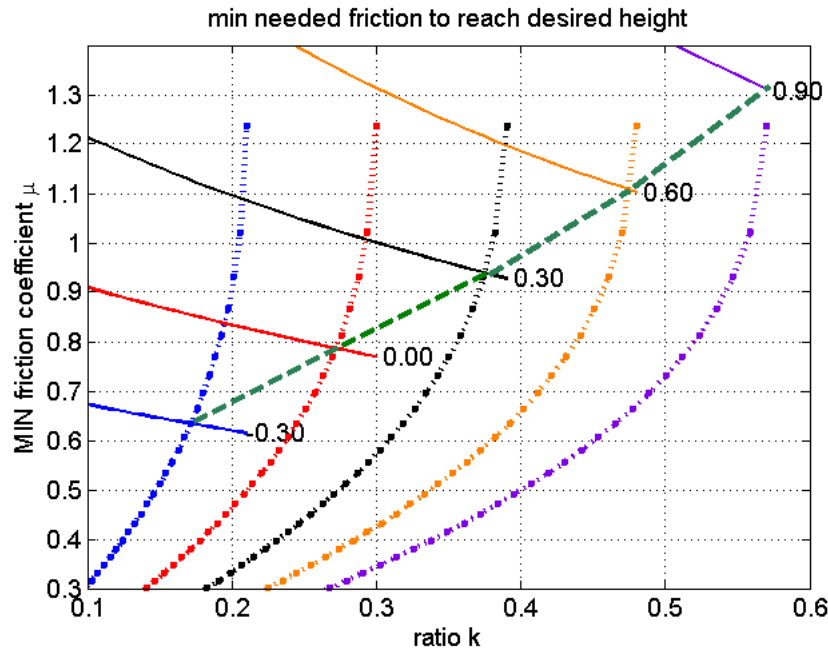


Figure 3.8: *additional static condition on μ : figure 3.5a is represented with the additional condition (3.16) in dashed-dotted lines; the green dashed line represents the reunion between the two conditions; a robot can climb on another and on several layers of robots if its design is situated above the green dashed line*

needed for climbing. By doing so however, the maximal geometrical limit of the robot's aspect ratio is also decreasing. From the perspective of climbing capabilities, a long robot with the CoM in the front is desired. On the other hand, one of the initial goals is to have as big of an aspect ratio as possible. One of the easiest modifications that can be done to the initial design without complicating the theoretical analysis, is adding retractable flippers (as shown in 3.10).

The robot could extend the flippers when climbing and retract them when waiting/ supporting other robots in a pile. It is therefore considered that a robot with flippers *occupies* the same place in the pile as one without, but *behaves* at climbing as one with lower aspect ratio and a changed relative position of the CoM. This is a significant step further in reconciling both of the aforementioned contradictory conditions

If it is assumed that the flippers are of inconsiderable weight compared to the rest of the robot, the CoM will remain in the same (absolute) position. The only things that will change compared to the original robot is the aspect ratio, and the relative position of the CoM. Thus, comparing a simple robot with a robot with flippers, for a same aspect ratio, one performs significantly better at climbing than the other, as shows figure 3.9

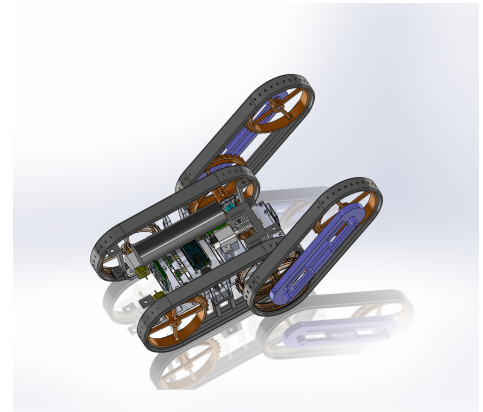


Figure 3.10: *example of robot with flippers*

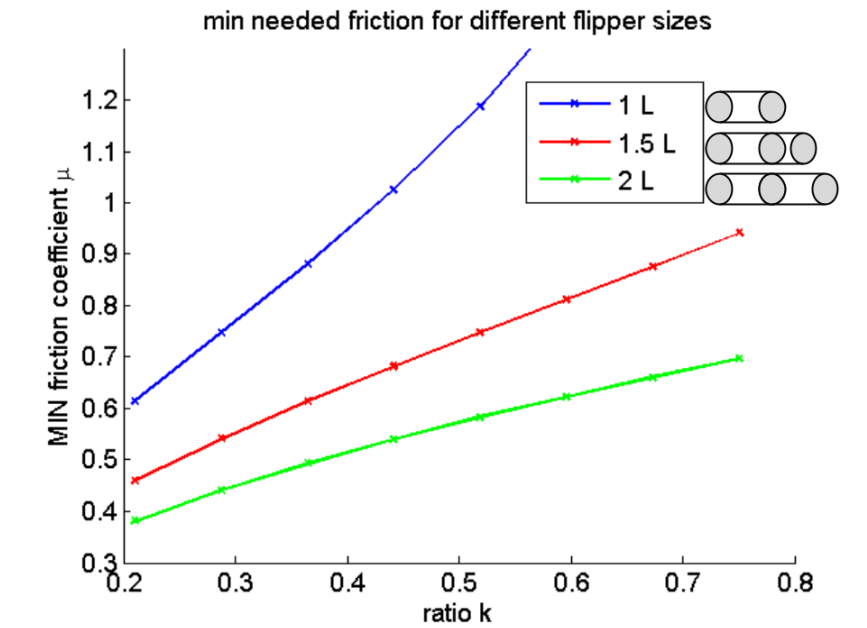


Figure 3.9: performance comparison between robots with different flipper lengths; green line represents flippers of the same size as the robot, which makes the new total length twice the initial one, whereas blue line represents no flippers at all;

The blue line of figure 3.9 is the same as the one from 3.5b. The other two lines represent robots with flippers.

Chapter 4

Mechanical Design

As the purpose of the robot is also to serve as a mobile building block, one desirable feature is **symmetry**. The overall shape has also to be as regular as possible to provide a stable support for the next robot. Size is not a constraint, yet the bigger the robot, the greater the manufacturing costs and 3D printing time. It is decided to keep the robot as small as the size of standard robotics components allow it.

Another desired feature for robust climbing is **high clearance**, in order to avoid as much as possible contact with other robots and objects other than with the tracks.

Succinctly, the robot is tank-treaded and has actuated flippers. The flippers have wheels at the end, which are independently actuated by another pair of tracks. A total of 3 motors are used: two for the track transmission, and one for the worm gear actuating the flippers. A commented overview of the whole assembly is proposed in 4.1. All parts except the deck are 3D-printed on a uPrint SE (Stratasys) out of ABS.

4.1 Tracks

The first components to be chosen are the tracks. Based on how well they grip and adhere, suitable motors have to be found and their size will dictate the dimensions of the rest of the robot.

Tamya tracks are low-cost, modular, and present good adhesion characteristics because of their protruded grousers and the material (soft rubber) they are made of. Furthermore, holes allow for very good fixation on a spur gear, which is a key feature in preventing them from slipping out in the cases of entanglement and shear effort.

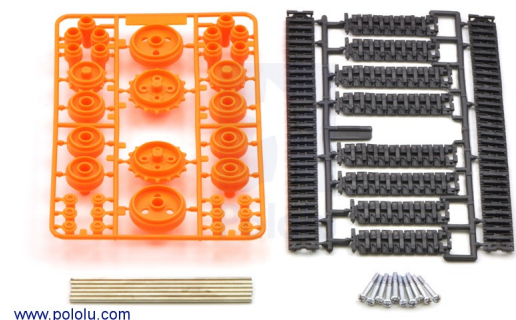


Figure 4.2: *Tamya track set from [43]*

4.2 Worm drive

Among other transmission possibilities between the motor and the flipper axle (belt, cable, chain, spur gears) , a worm-drive is chosen because of its **compactness** and **self-locking** capability. The motor can

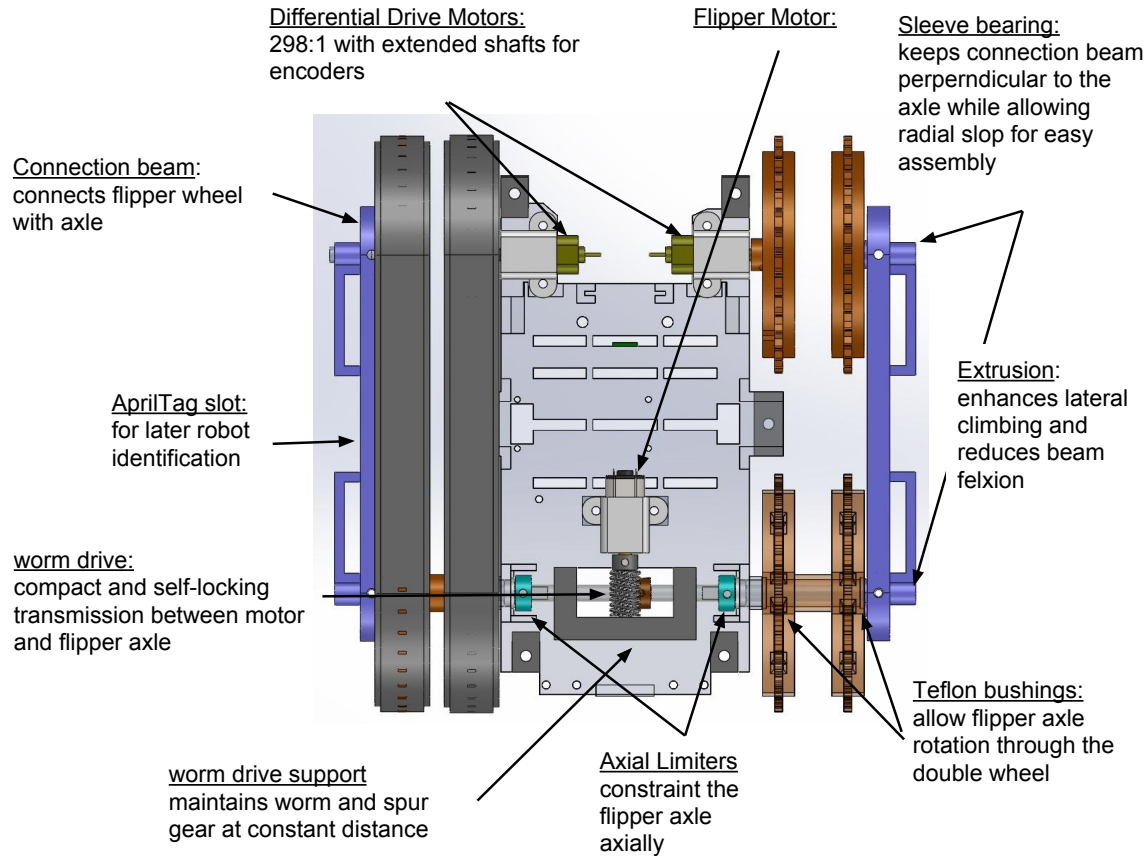


Figure 4.1: commented overview of the robot assembly

be placed adjacent to the axle, occupying a minimum of space.

The key feature to be considered when choosing the worm drive is non-backdrivability, or the self-locking capability. It is approximately correct to say that for a worm drive to be self-locking, the same law for an object on an inclined surface applies.

$$\mu > \tan(\theta)$$

where μ is the friction coefficient between gear materials and θ the lead angle of the worm gear. The worm gear is out of stainless steel and will be mounted on the motor shaft, whereas the spur gear on the flipper axle is made out of bronze. In this case, the coefficient of friction between bronze and steel is ≈ 0.16 which is more than twice than the lead angle of the worm gear ($7^\circ 7'$).

For speed consideration, a low transmission ratio is chosen. The spur gear has 30 teeth, and double thread, which means the **transmission ratio = 15**.

4.3 Motors

4.3.1 Wheels

One of the most determinant components for the design of the chassis, and hence for the whole robot is the size of the motor. The motors are dimensioned based on the model and computations presented in chapter 3. As a results of (3.15) the necessary torque for each motor is :

$$\begin{aligned} M_{tot} &= M_1 + M_2 = F_1 \cdot R + F_2 \cdot R \\ &= \frac{\mu \cdot mg}{1 + \mu^2} \cdot R + \frac{\mu^2 \cdot mg}{1 + \mu^2} \cdot R \\ &= \frac{\mu(1 + \mu) \cdot mg}{1 + \mu^2} \cdot R \end{aligned}$$

with M_1 the back wheel torque, and M_2 the front wheel torque and R the wheel radius.

Figure 4.3 illustrates the torque for each wheel needed to maintain the robot in static equilibrium given a certain friction coefficient. The rolling coefficient is assumed non-existent. As one track connects the two wheels, obviously only one motor is used, and the total torque that it should provide is shown as well.

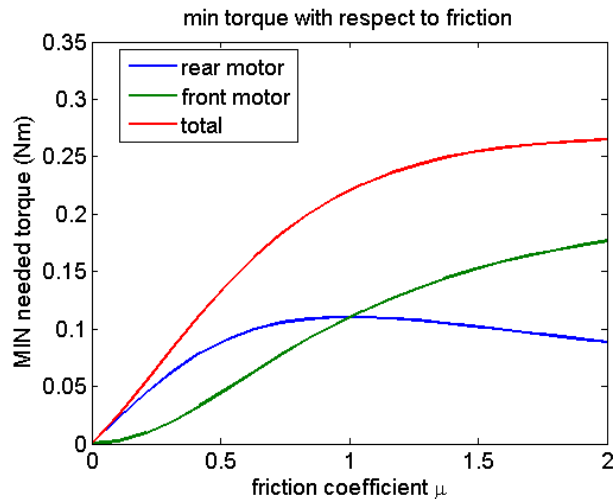


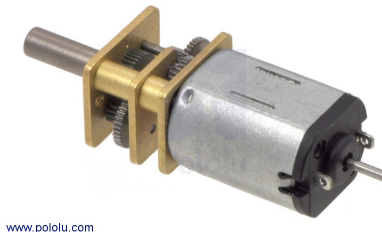
Figure 4.3: with a mass of 0.5 kg, a radius of 4.5 cm and a friction of 1.8

For a given friction μ , there exists an aspect ratio and a position of the CoM that makes climbing possible, as has been shown in the previous theoretical analysis. Hence, a reliable μ has to be chosen and the motor torque has to be computed accordingly.

Rubber on rubber has a friction coefficient of about 1.1, but when the tracks are aligned the grousers grip very well, being able to maintain friction on a ramp of more than 60° . In this case, a friction coefficient of ≈ 1.8 is chosen for the torque computation, as it gives also a satisfying safety-factor.

From figure 4.3 it results that a motor with a nominal torque of around **0.25Nm** is desired.

The Pololu high power 298:1 Micro Metal Gearmotors have this characteristic while coming in a very



www.pololu.com

Figure 4.4: *298:1 HP Micro Metal Gearmotor with extended shaft for encoders* <http://www.pololu.com/product/2208>

compact form. The integrated gearbox simplifies the design, as the robot wheel can be directly mounted on the motor output axle. An extended shaft is also a comfortable option, and avoids a custom designed encoder. Adequate motor brackets facilitate attachment to the chassis.

The main specifications are listed in the chart below 4.1

Motor type	Stall Current (6V)	No-Load Current (6V)	No-Load Speed (6V)	Stall Torque
high-power	1.6 A	0.07 A	100 RPM	0.5 Nm

Table 4.1: *motor specifications*

4.3.2 Worm Gear

Two options are available: either a servo motor, or again a DC motor with encoder. In order to recover the robot after an overturn, it is necessary to have flippers move 360° , which is difficult with a servo (usually limited to only 180°). Moreover, the servo output shaft has a non-standard design and is very difficult to find one that fits to the desired worm gear.

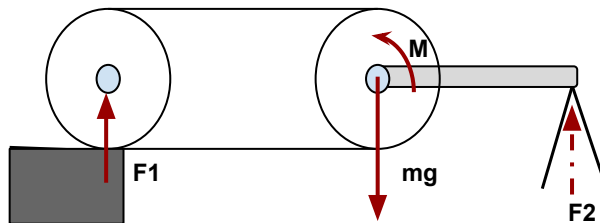


Figure 4.5: *forces exerted in an extreme situation: the full arrows represent the efforts on the robot body (excluding the flipper)*

For a rough estimation, it is considered that the robot has to maintain its own body weight horizontally with the CoM located on the front wheel, as shown in figure 4.5. With a weightless flipper, the static equations for the robot body would yield:

$$M = \frac{mg \cdot L}{2}$$

For a weight of 0.5kg and a length of 10cm, $M = 0.25 \text{ Nm}$. This however stays a very optimistic estimation, as it neglects the friction force between spur and worm gear, which can be considerable in a case of bad assembly. The transmission ratio of 15 will however allow for enough safety margin.

Hence, the same choice (298:1 HP motor) as before is done for the worm gear motor too.

4.4 Chassis

From the size of the motor and of the worm drive, the overall size of the chassis can be established. The alignment of the track wheels and the position of the worm gear motor are essential features to pay attention to. Figure 4.6 explains the key design details.

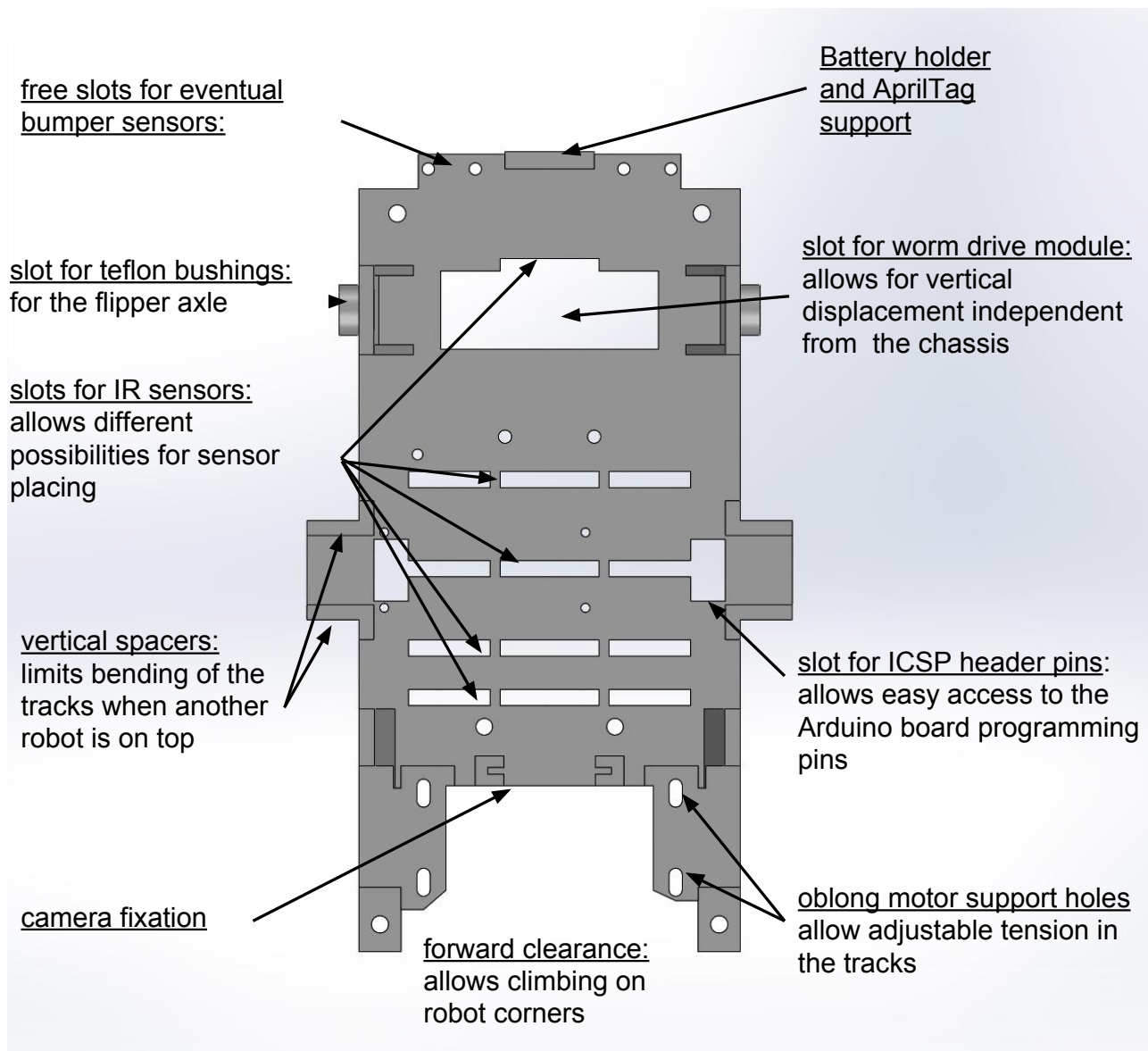


Figure 4.6: robot chassis with main characteristics

4.5 Worm Drive Support

It is essential to ensure that the contact distance between worm and spur gear will stay within controlled limits, or else gear unmeshing might occur when the torque increases. Therefore, a support is designed to hold the axle and the motor at constant distance. The part is independent from the chassis and a slot in the latter allows for it to move vertically while constrained horizontally and laterally.

An illustration is provided in figure 4.7

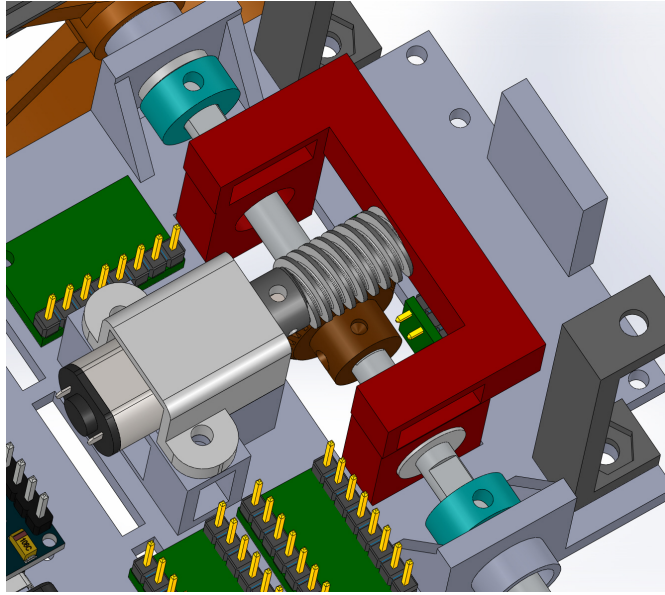


Figure 4.7: *the worm drive support - illustrated in red*

4.6 Wheels

Based on the pitch diameter of the wheels (**0.477**) included in the Tomyia track pack, custom wheels are designed. The robot has 3 pairs of slightly different wheels. The **motors wheels** are provided with a set screw for fixation on the motor, whereas the **flipper wheels** spin freely around a 4-40 screw spacer. The screw head and a nut at the other end constrain them axially.

The **double wheels** ensure the transmission between the motor and the flipper wheels. They are also provided with Teflon bearings, allowing the flipper axle to turn independently through.

4.7 Deck

In order to stay faithful to the theoretical model, the deck should not be higher than the wheels. If this happens, this will act like a rail, guiding forwards the next robot that will climb. This effect is undesired, because by guiding the tracks small maneuvers on top of a robot become even more challenging. It would basically act as a small docking/alignment mechanism, as in modular robots designs, which has to be avoided.

The deck is a simple flat surface with a pattern on it allowing the robot on top to localize its position when coming from the rear. The longitudinal stripe is for alignment and line following, whereas the

transverse one is for edge detection and counting. As the tracks are also black, the IR sensors would receive relevant information when on top of them.

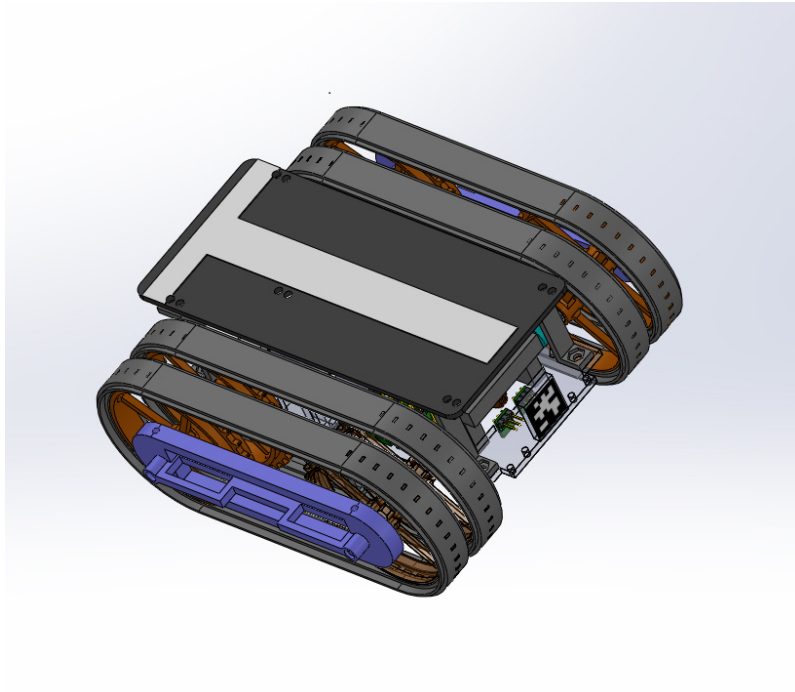


Figure 4.8: *robot deck with longitudinal and transverse white stripe for line following and edge detection*

It is laser-cut in a 2.5mm thick acrylic sheet, allowing fast and low-cost replications. Electrical duct tape is used for the black pattern and normal paper for the white stripes. The latter are 2cm thick, matching the size of the IR sensor slots in the chassis (figure 4.6).

Chapter 5

Embodiment and Climbing Assessment

For the first model to be built, a conservative aspect ratio of 0.37 has been picked. The robot is meant to be able to climb on random obstacles (books, boxes) which do not offer a very high amount of friction (≈ 0.5). Its purpose is also to see what problems might arise from the interaction between two robots. Some pictures of the embodiment are shown below (figure 5.1).

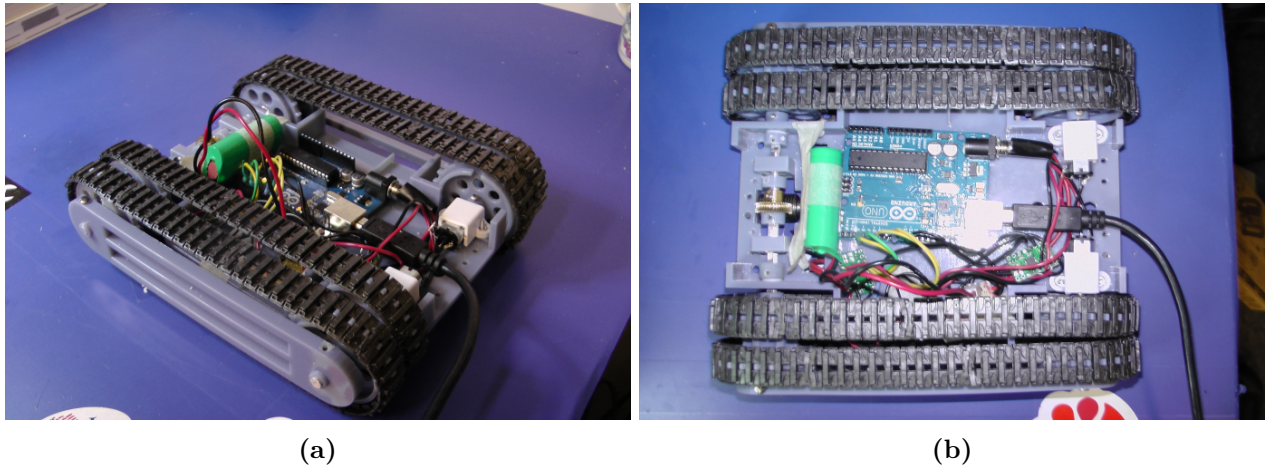


Figure 5.1: *1st model built with a prudent aspect ratio of 0.37; only the minimum electronics for remote controlling are present; the flushed frontal design of the chassis does not allow it to tackle obstacles diagonally*

The robot has a height of around 52mm, an inter-axis length of 140mm a width of 170 mm, a mass of 441g and an offset of the CoM $d = 0.16\%$. The latter is determined by holding the robot by a string attached to one of its coners and observing how the posture of the robot is affected by gravity. This is done from two different corners, the angles are mesured, and d is obtained using straightforward trigonometry

The robot can climb on books more than twice its height ($>11\text{cm}$), and has no problem climbing on a fellow robot. When coming at a 90° angle to another robot, it can also climb without flippers. In this configuration, 16 trials have been conducted, where a robot has to climb on another. The robots are manually placed and are either facing each other or at a 90° angle. No failures have been recorded.

When trying to climb diagonally (with an angle around $30\text{-}70^\circ$) however, the robot would get stuck

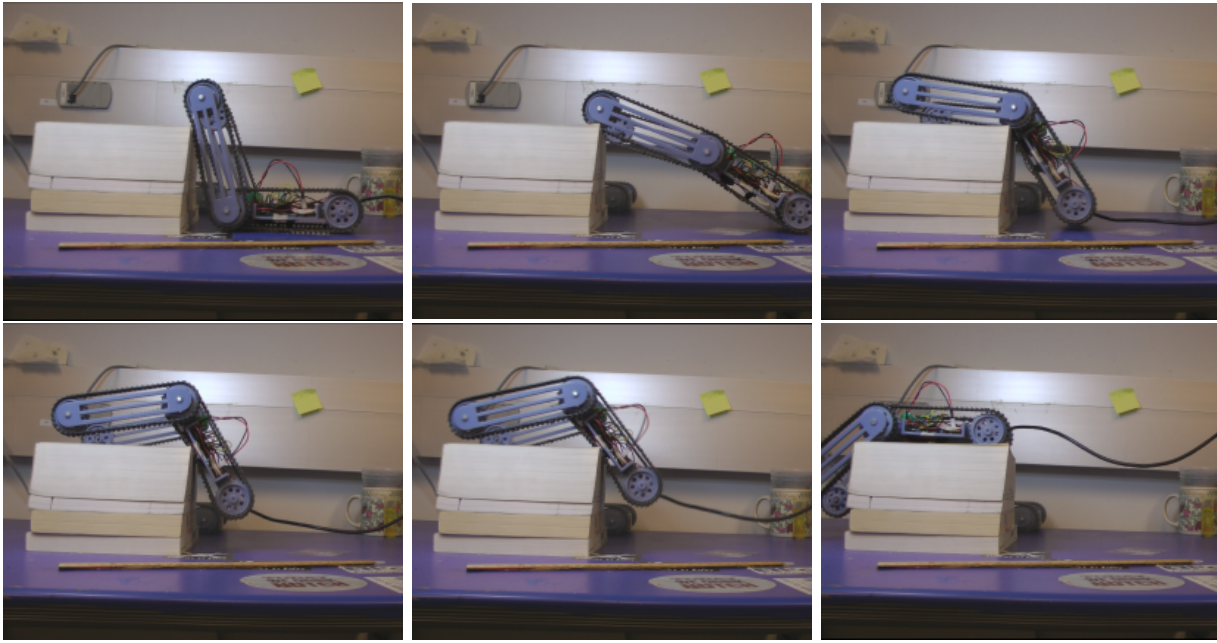


Figure 5.2: *illustration of climbing on an obstacle more than twice the height of the robot*

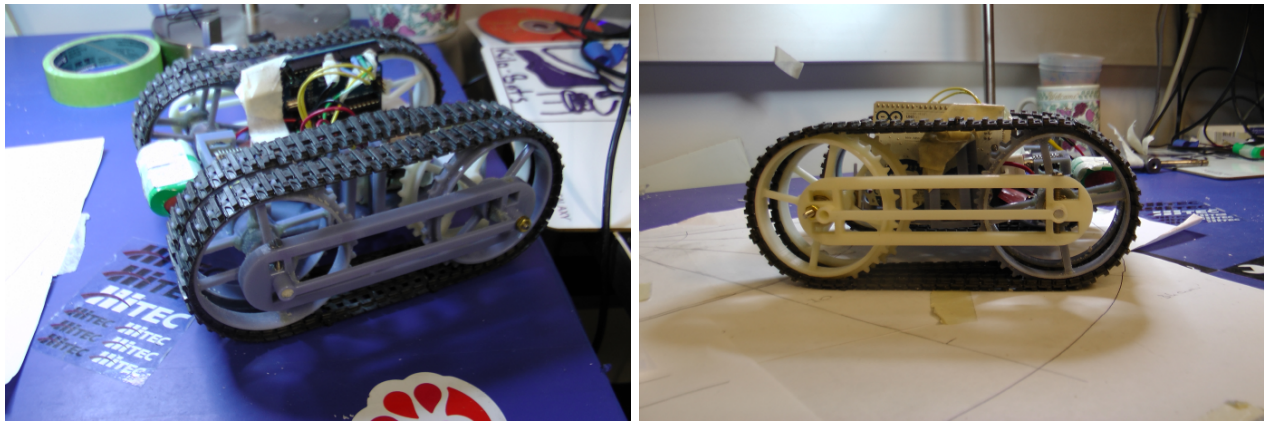


Figure 5.3: *second embodiment with an aspect ratio of $k = 0.7$*

on the corner. The flippers can be used to push out of the position, yet it does not constitute a satisfying solution. To simplify subsequent sensing and planing, the robot should be able to climb without getting stuck.

Besides failure on the corner, the robot aspect ratio has been too conservatively chosen, which calls for a redesign. A **second model** is build, with a double aspect ratio, and a more thorough assessment is done. Emphasis is also put on having a high clearance from the ground and from the front. Clearance from the front (as shown in 4.6) help in tackling with corners and clearance from the ground avoids situation where the robot would get stuck on the edge of another, as shown in figure 5.5. This latter design is the final one and is the embodiment of the CAD schematics shown previously.

The characteristics of the both robots are shown are detailed in chart 5.1. Figure 5.4 situates the realized robots in the theoretical analysis, while giving an idea of their performance relative to each other.

The difference in needed friction with and without flippers for the first design can be easily noticed. It can also be observed that while doubling the aspect ratio, the necessary μ increases only from ≈ 0.5 to ≈ 0.7 .

design	aspect ratio	height (mm)	L_{total} (mm)	$L_{interaxis}$ (mm)	width (mm)	CoM (% of length)
1st	0.37	52	183	140	170	16
2nd	0.7	74	176	105	180	64

Table 5.1: robot specifications

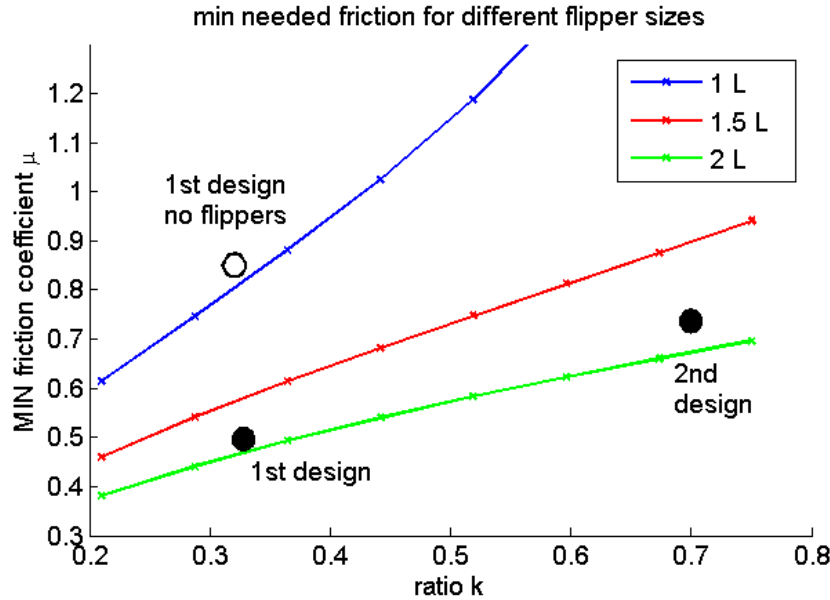


Figure 5.4: situation of the 2 designs with respect to the theoretical predictions; 1st design is represented with open and closed flippers, while the second only with open flippers (for visibility reasons)

5.1 Climbing Assessment

The goal of this section is to prove that the second robot design is capable of robustly climbing on identical fellows, and that it constitutes a solid platform for any further or related work on construction through self-assembly. Independently of its possible application, it is necessary to prove that climbing and mechanics will not constitute an issue.

5.1.1 Phase I

In a first phase, the robot has to climb on another fellow from 10 different angles, and for each angle 10 trials are conducted. This is first done with the flippers forwards, second with the flippers forwards but with additional weight on them (100g), and finally with the flippers backwards. The robot always starts from a distance of 34cm to the other one (center-to-center), with a different angle α , as shown in 5.6.

The results are shown in the charts from 5.7. For each case, there are three possible outcomes: success (blue), failure (red), or misalignment (orange). backwards is declared when the robot ends on top of the

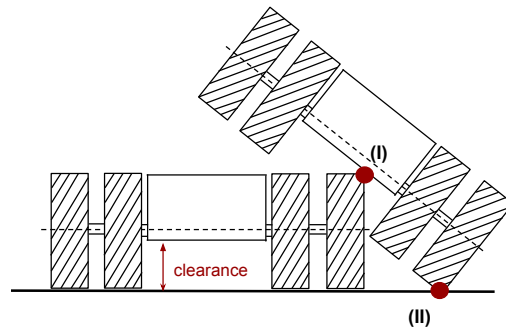


Figure 5.5: *ground clearance and side stuck position: most of the robot's weight is supported by red point (I), which means there is not enough weight on (II) to ensure traction*

other, and remains in a stable position, horizontally. Failure occurs when the robot does not make it to the top at all (sliding, or being stuck). Finally, “misalignment” means the robot has climbed, but does not stop in a stable horizontal position, by for instance sliding on the edge of the robot beneath (ending a situation similar to the one from figure 5.10).

Due to the high aspect ratio, it can be observed that climbing with the flippers forwards is almost impossible when the robot is not heading perpendicularly towards the other one. With weight on the forward flippers, this can be solved. However, this is not a satisfying solution, as the weights on the flipper destabilize the robot during deployment and retraction, leading sometimes to its falling down from the structure.

Driving with the **flippers backwards** offers the advantage of significantly shifting the CoM of the robot forwards, while keeping most of its climbing capabilities, as can be seen in figure 5.6. Furthermore, the flippers can be manipulated without any loss of balance. This configuration will be kept for the rest of the work.

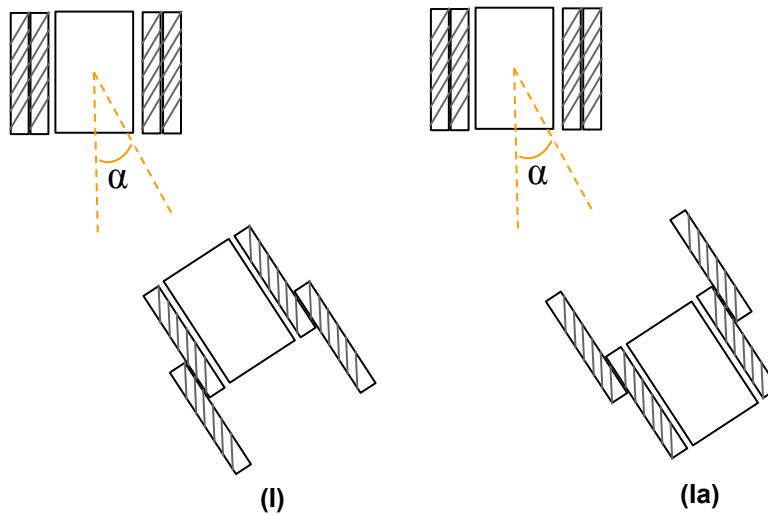


Figure 5.6: *different climbing configurations: with flippers backwards (I) and flippers forwards (Ia);*

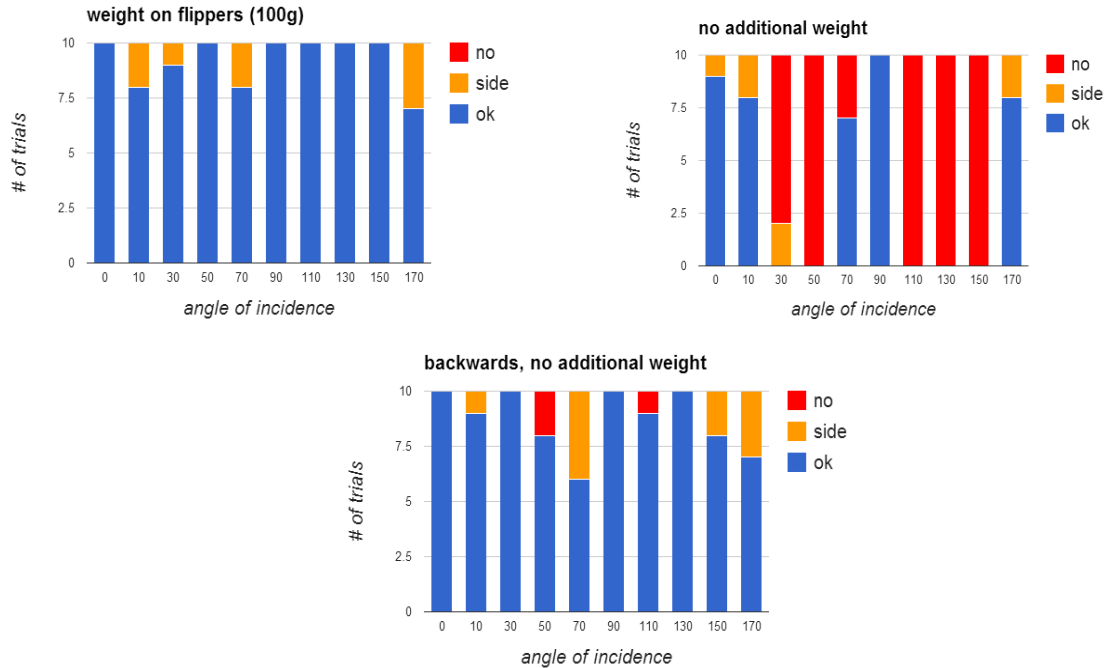


Figure 5.7: results in climbing from different angle in different configurations; 10 trials have been conducted for each angle (spanning from 0 to 170°); successes are indicated in blue, failures in red, and slides in orange; the backwards configuration is kept as definitive

5.1.2 Phase II & III

In a second phase, climbing from a shifted position is assessed, for different offsets, 10 trials each time. Finally, a third setup assesses climbing on two layer of robots. The robot on the second layer is set up with a different angular offset with respect to the rest of the structure, as shown in 5.8

The purpose of the second phase is to demonstrate that even severe misalignments preceding climbing do not lead to a position where the robot would get stuck. The Robot has a total width of around 18cm, and the most difficult situation occurs when the offset is close to half of its length (8 - 10 cm). In this situation, the configuration of the robot resembles the one sketched in 5.5. Because of the symmetry of the robot, a contact point near the sagittal plane will result in a loss of traction force, and to a stand-still of the robot despite the turning tracks. Two factors contribute to avoid this problem: the ground clearance and the width of the robot. However, for symmetry reasons, the width has to be close to the length, and due to the position of the robot wheel axis, the maximum possible clearance is also limited, which leaves very few degrees of freedom for more changes.

Finally, the third phase is meant to show that climbing can be performed on two robot layers independently of the last robot's orientation. Results from the right figure 5.9 are also divided in "success" (blue), "side" (orange) and "misalignment" (grey). In the first case the robot climbs to the top and stops in a stable position. Idem in the second, but this time it slides on the side of the last robot (similar to the position illustrated in 5.10. In the third case, after climbing half the way, the robot is not properly aligned

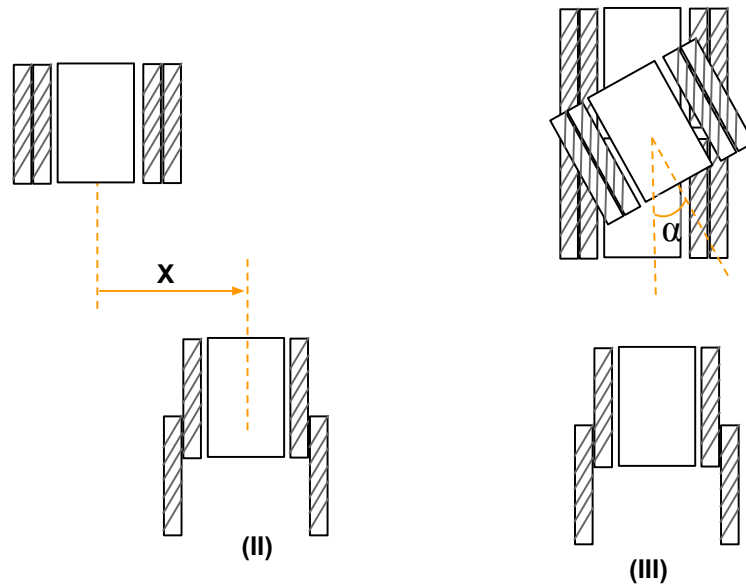


Figure 5.8: *phase (II) is meant to assess how often the robot would get stuck on the side (as sketched in 5.10 and phase (III) is meant to assess the performance of climbing on a 2-layered structure*

anymore, and slides before reached on the second level. This is not considered a failure, as it constitutes more of a sensing than a climbing problem.

To conclude with, if the robot is to drive with the flippers backwards, an aspect ratio of 0.7 presents satisfying climbing robustness. Given the occasional failures however, it would certainly not be wise to push it further.

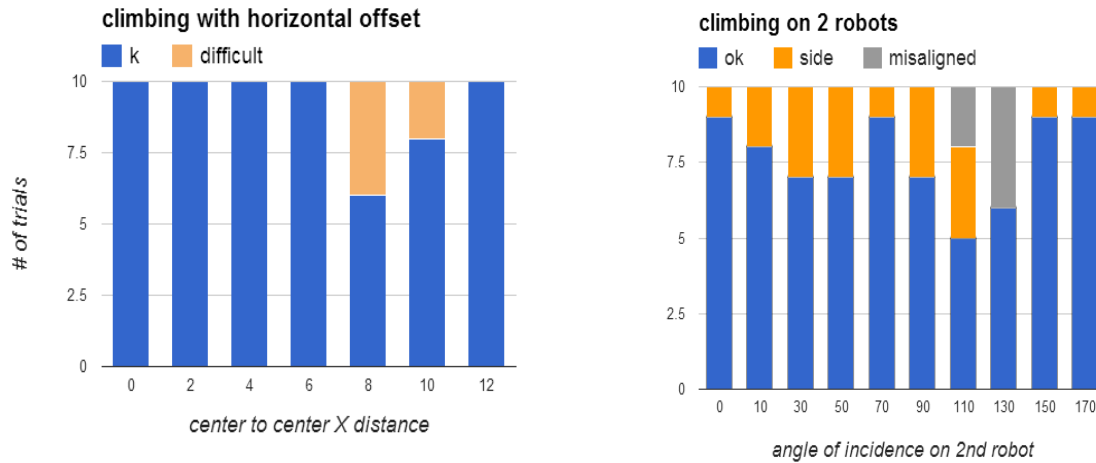


Figure 5.9: results of phase II (climbing from a shifted position) and of phase III (climbing on 2 robot layers); in phase II, “difficult” means the robot gets temporarily stuck in a position similar to 5.10, and in phase III “misalignment” occurs when the robot falls before the second layer is reached

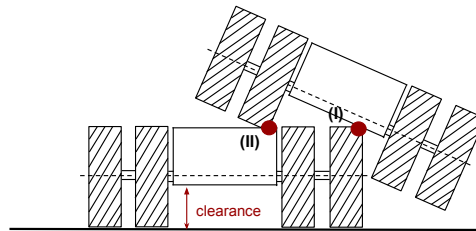


Figure 5.10: sliding on the edge of another robot can occur after climbing; although the robot is not stuck, the climbing has to be repeated, because it is not considered a stable position for further construction

Chapter 6

Towards Full Autonomy

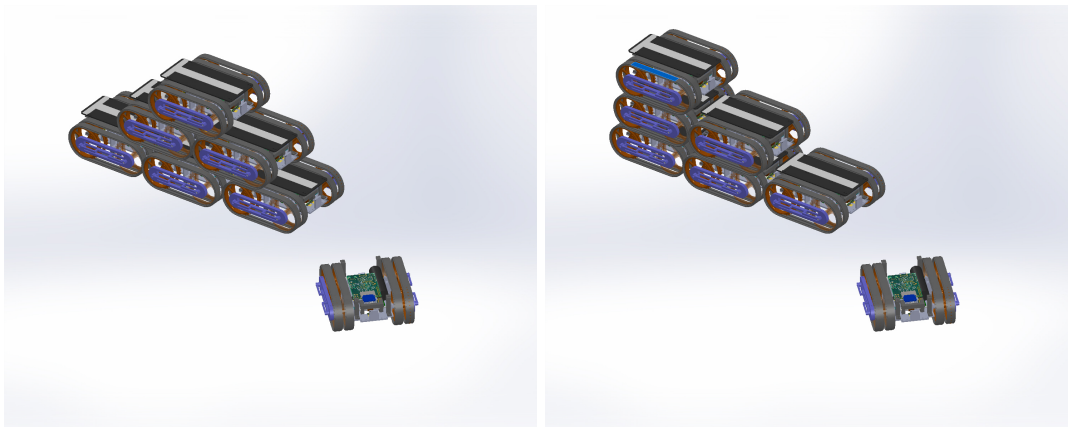
In the previous chapters, the construction of a climbing robot with flippers has been detailed and justified. An optimal size for the robot has been chosen based on mathematical considerations and the climbing capabilities have been assessed. The results have shown that climbing on an identical fellow can be done in a robust manner and in a multitude of configurations.

The robot has now all the needed physical characteristics to be involved in the *autonomous* construction of a structure.

For the following steps, the goal is to pick a type of structure and implement an algorithm that would lead to its autonomous assembly/construction.

6.1 Choosing a Structure

Social insects can “self-assemble” to form rafts, chains, bivouacs, towers, and bridges. In our case, a structure has to be chosen that demonstrates both the climbing and self-organization characteristics of the robots. The possibilities span from 2D and 3D pyramids to ramps, and cantilevers. Furthermore, the structures can be stand-alone (made only out of robots) or include objects from the environment.



(a) 2D pyramid, or tower

(b) 2D ramp

Figure 6.1: *some possibilities of self-assembled structures; the 2D tower is chosen*

It is decided that if the robots are capable of building a **stand-alone 2D tower**, they would necessarily be equipped with most of the necessary sensors to achieve more complex structure building in 3D. Furthermore, it would also allow the demonstration of an algorithm scalable to an arbitrary-sized structure. The same kind of algorithm could then easily be extended to involve also random objects from the environment.

6.2 Building a 2D Tower

6.2.1 Challenges

Building a stand-alone 2D tower implies solving the following issues:

1. **localization & navigation** : find the position relative to other robots / to the structure and navigate to them
2. **alignment** : find the orientation relative to other robots and go to a favorable orientation for climbing
3. **climbing** : decide on starting and ending point for climbing
4. **positioning** : find which place in the structure is to be occupied and go to it

Furthermore, because of the mechanical aspects of the tracks, the robot has severely limited maneuverability when on top of another one. High track-on-track friction as well as the grousers on the tank-treads prohibit big rotations but practical experience shows that they allow for small maneuvers, such as for the ones required in **line following**.

This means that all orientation issues have to be solved **prior** to climbing. In other words, the robots have to be as well aligned as possible before starting climbing, as this will directly influence their capability of reaching the desired position in the structure.

6.2.2 Solutions & Sensors

To deal with issue 1 and 2, robots in literature often use IR emitters and receivers. Docking systems are an example of such applications. The Roomba robot (iRobot) uses a 360° lens and a docking station with 2 IR beams to find its way back and dock. A similar system is detailed in [44]. Although this may work with a single robot and one docking station, it is difficult to see how this solution would scale to a multitude of robots, and how all the IR emitters and transmitters would have to be mounted on the existing mechanical design in order to achieve this.

Moreover, a careful electrical design would be necessary to encode the IR on different channels, filter noise, and avoid interferences. One last drawback would

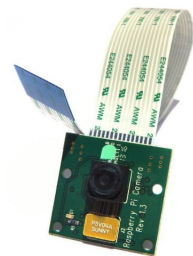


Figure 6.2: a Raspberry PI camera is used for navigation and alignment

be that the robots in the structure will have to continuously emit IR light (and consume battery).

A **camera** avoids all of the aforementioned issues. Cameras have become more and more accurate and affordable. A Raspberry Pi camera 6.2 costs below 30\$ and has a resolution of 5MP. However, markers have to be used and image processing often comes with at a heavy computational expense. The computational power can be provided by an Raspberry PI, model B (BCM2835 chip, 512 RAM, 700 MHz, running Linux), which provides a compact, low-cost and user-friendly way to exploit the camera's capacities.

To deal with issue 3, **an accelerometer** (LSM303DLHC 3D, Pololu) is used. Based on the gravity force vector, the robot can extract the angle and infer if it is in a horizontal position or not. The robot can know when to start and stop climbing, and can use its flippers to reposition if it is turned over.

Finally, in order to deal with issue 4 a combination between IR sensors, motor encoders, camera and accelerometer will be used. The **IR sensors** (QTR-L-1A, Pololu) will provide the robot with guidance when on top of another one (figure 6.3), and the **motor encoders** (Optical Encoder Pair, Pololu) will provide relevant information on the travelled distance.

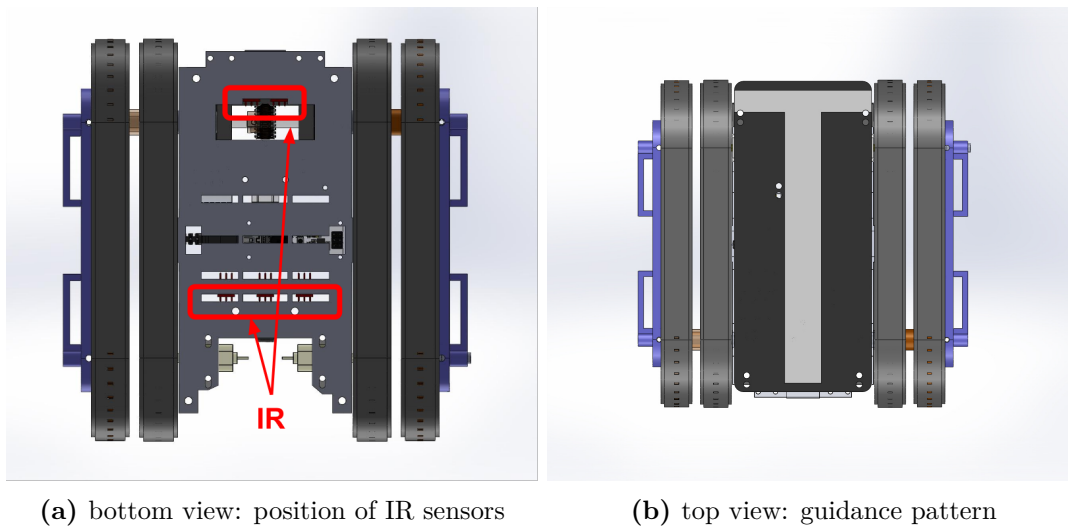


Figure 6.3: *bottom and top view of the robot; when on another robot, the robot mostly relies on its IR sensors; the longitudinal white pattern guides the robot and the transverse one indicates an imminent edge*

Because of the high clearance to the ground(27mm) that the robot has, the IR sensors have been picked such as to give reliable results at a distance of at least 0-20mm. However, such a relatively big distance makes measurement much more sensitive to noise, and especially shading.

The motor encoders have been chosen such as to be mounted on the extended shaft of the motor, in a compact way without additional custom design.

6.2.3 Electronics - Overview

The algorithms and behaviors are programmed on a Arduino Micro (atmega32u4), and all the sensors are connected to it, except the camera that is connected to the Raspeberry PI (RPI). The latter communicates serially with the Arduino.

Other than the already mentioned sensors and chips, 2 motor drivers (DRV8833, Pololu) are used to power the motors. A LiIo battery (7.2V, 660mAh) supplies the drivers with power. An Anker©Astro Mini portable charger (5V, 3000 mAh, 1A output) is used to power the RPI and all other devices.

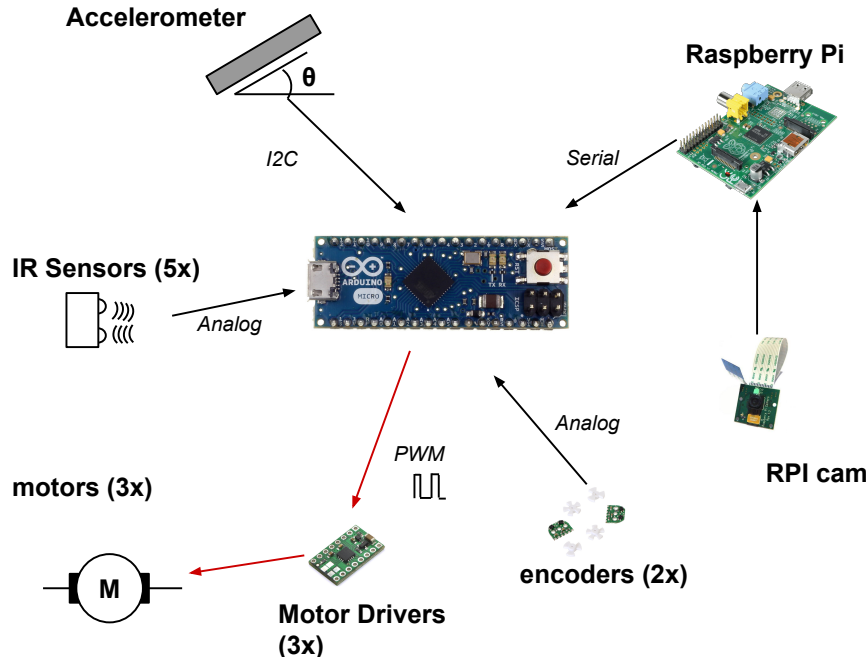


Figure 6.4: *overall schematic of the robot's electrical components; black arrows are inputs and red are outputs; the communication protocol is indicated in italic*

For debugging purposes, a PGM01A programmer, providing serial communication with the Arduino, is used.

An overview of the electrical components and their placement in the robot is given in figure 6.5. Figure 6.4 proposes an overall electrical schematic of the robot and indicates the communication protocol each sensor uses.

6.2.4 Solving the Alignment Problem

As already emphasized, alignment is essential for achieving a structured construction. The robot has been purposely designed without mechanical alignment features, in order to avoid lattice-like structures. The camera is thus the most suitable for such a system, allowing flexibility, and a wide span of operations while having only small impact on the mechanical design.

Extracting position and orientation of other robots through image processing alone is not trivial. One of the easiest way to do it is using fiducial tags and use appropriate mathematics to extract position and orientation. In our case, 3D orientation is extracted using **April Tags**.

In [45], a method for robustly extracting April Tags from an image is described. Edges are detected by computing the magnitude and direction of adjacent pixels. The results are then clustered together

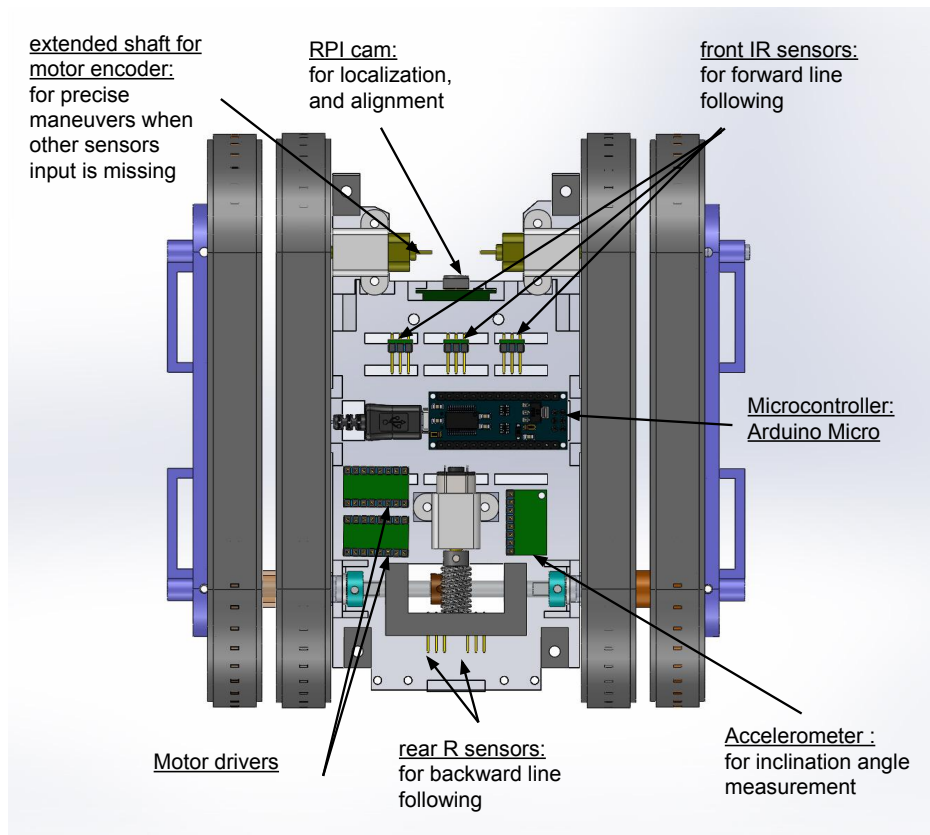
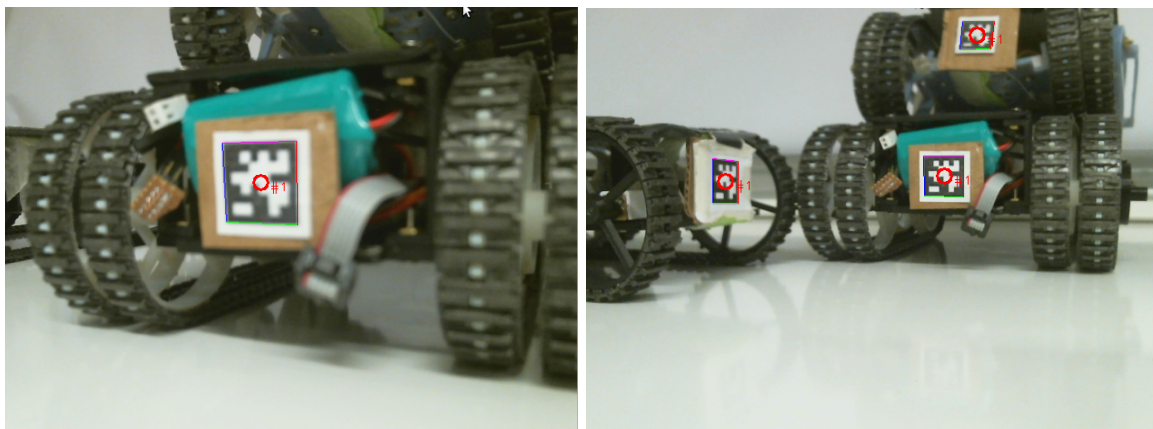


Figure 6.5: overview and placement of the sensors in the robot; the RPI board and batteries are omitted for visibility purposes



(a)

(b)

Figure 6.6: example of April Tag detection: for visualization purposes, a rectangle is drawn around the tag when detected; orientation information is then extracted using homography

and line segments are fit using a least-square procedure. All 4-sided shapes (named “quads”) are then extracted. Figure 6.6 illustrates the result of the extraction algorithm.

Finally, given the size of the Tag, the distance to it can be computed, and an isomorphism called *homography* is used to extract the 3D orientation. Using C++ libraries available on-line ¹, the code is implemented on the Raspberry PI. April Tags are placed on the back of the robot and the camera is added in the front, as shown in 6.7. Later on, AprilTags should also cover the sides of the robot.

The April Tag detection algorithm works on the RPI at around 2.9 fps.

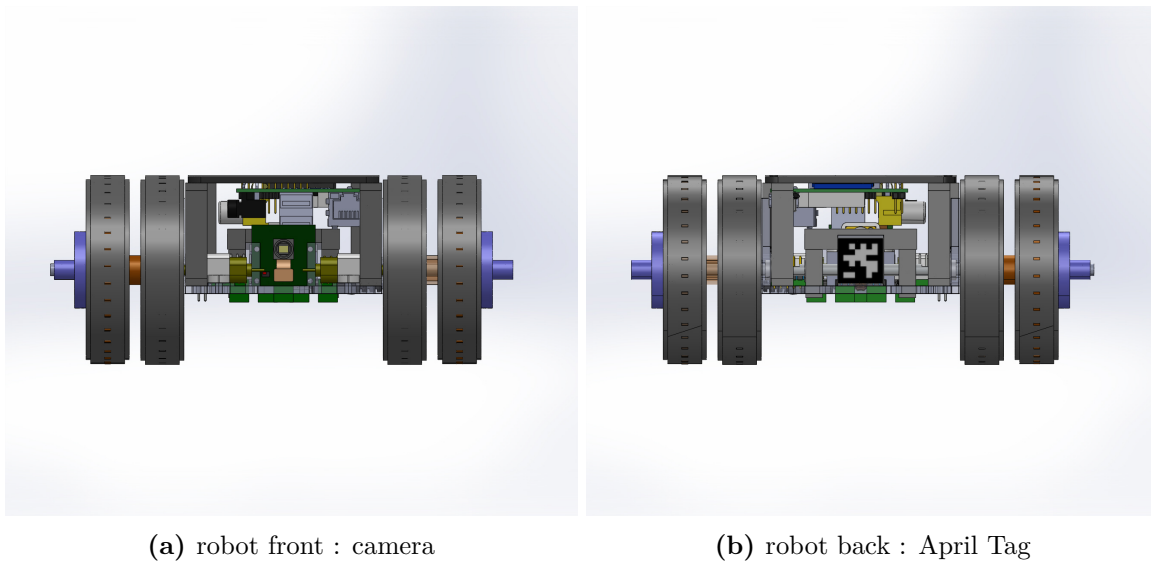


Figure 6.7: *position of the camera and of the April Tag on a robot*

6.3 Construction Algorithm

In order to achieve a robust algorithm for self-assembly, and given the context of collective robots, the following criteria are compulsory:

- the algorithm should be independent of the size of the structure
- the algorithm should be the same on all robots

Two reasonable ways of building a 2D tower with 6 robots are listed below.

Each robot has to follow the steps that have been already cited in section 6.2.1: localization, alignment, climbing and positioning in the right spot. The second algorithm has to be excluded, because it requires a prior knowledge of the final length of the structure in order to build a base of adequate length. In contrast to that, the first algorithm will keep on incrementing along the initial 3 positions, without the need of building a base of a specific length. Moreover, it reduces the travel distance of a robot on a layer of other robots, which is a big plus.

A very general pseudo-code is shown below:

¹<http://people.csail.mit.edu/kaess/apriltags/>

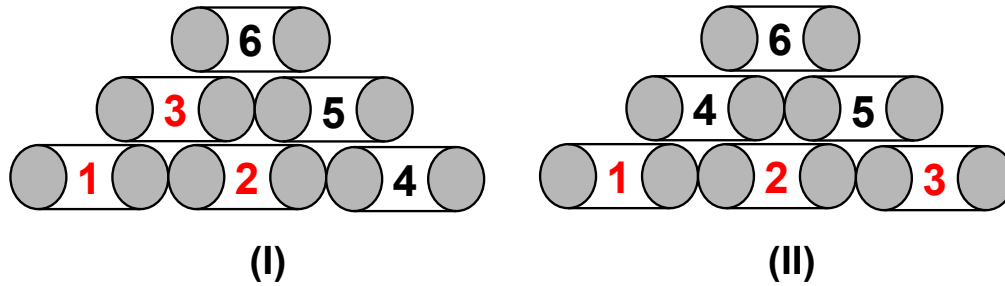


Figure 6.8: possible orders for a 2D tower with 6 robots: the first 3 steps are highlighted in red for better visibility

```

while (1) do
  if robot seen then
    align and climb
  else if climbed and horizontal again then
    if robot seen in front then
      align to it, don't climb; break ;           ▷ position 5
    else
      count number of robots beneath
      if 2 robots beneath then
        go in the middle of the two underlying robots; break;   ▷ position 3, 6
      else if 1 robot beneath then
        go align behind it; break;                       ▷ position 2, 4
      end if
    end if
  end if
end while

```

The comments on the right indicate when the robot exits the algorithm. The position number applies to the order from figure 6.8. An illustration of all the steps of this algorithm for a 6 robot pyramid structure is shown in figure 6.9.

6.3.1 Implementation

A global structure formation strategy has been given. The implementation is done while applying the criteria below. It should:

- limit as much as possible maneuvers on top of other robots (high track-on-track friction makes maneuvering unreliable and unsafe)
- limit open-loop maneuvers (a lot sliding and slipping occurs, which biases the motor encoder information)
- provide additional safety by doing redundant sensing (i.e. more than one sensor is used to detect the same feature)

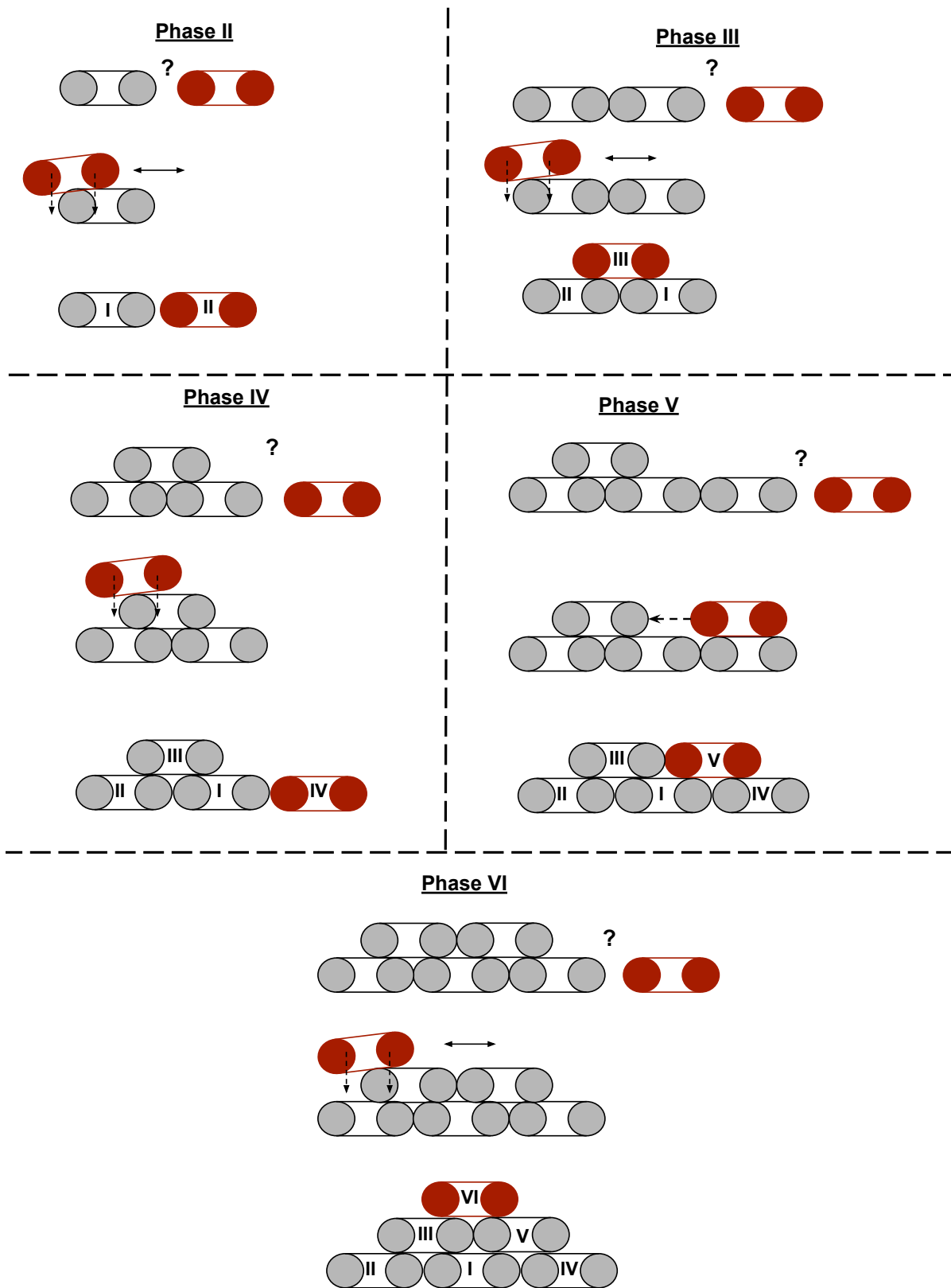


Figure 6.9: illustration of an arbitrary-sized structure formation algorithm for a 2D pyramid; the first 6 steps are shown; generally, the robot has to climb and count the number of robots beneath; from an algorithmic perspective, phase II is the same as phase IV, and III the same as VI; in phase V counting is not necessary as an April Tag is detected directly after climbing; this algorithm can be reiterated for an arbitrary-sized structure and stays the same for each robot

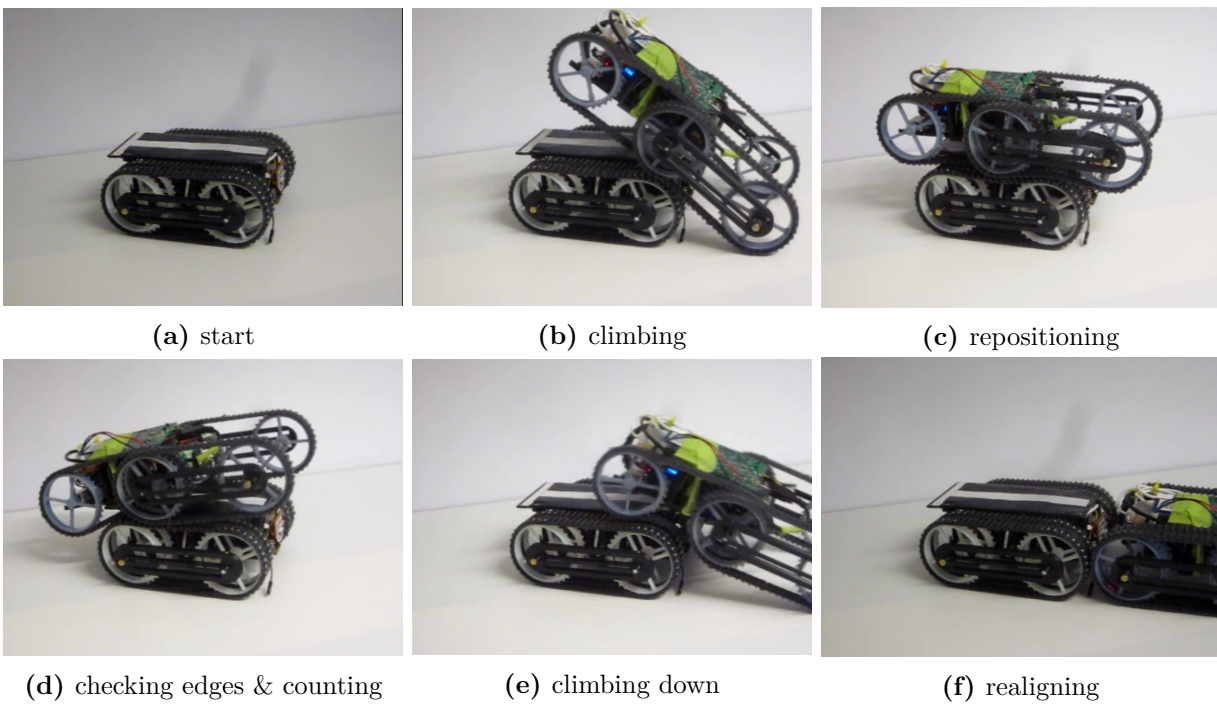


Figure 6.10: recording of phase II; the exact same strategy is iterated in phase IV

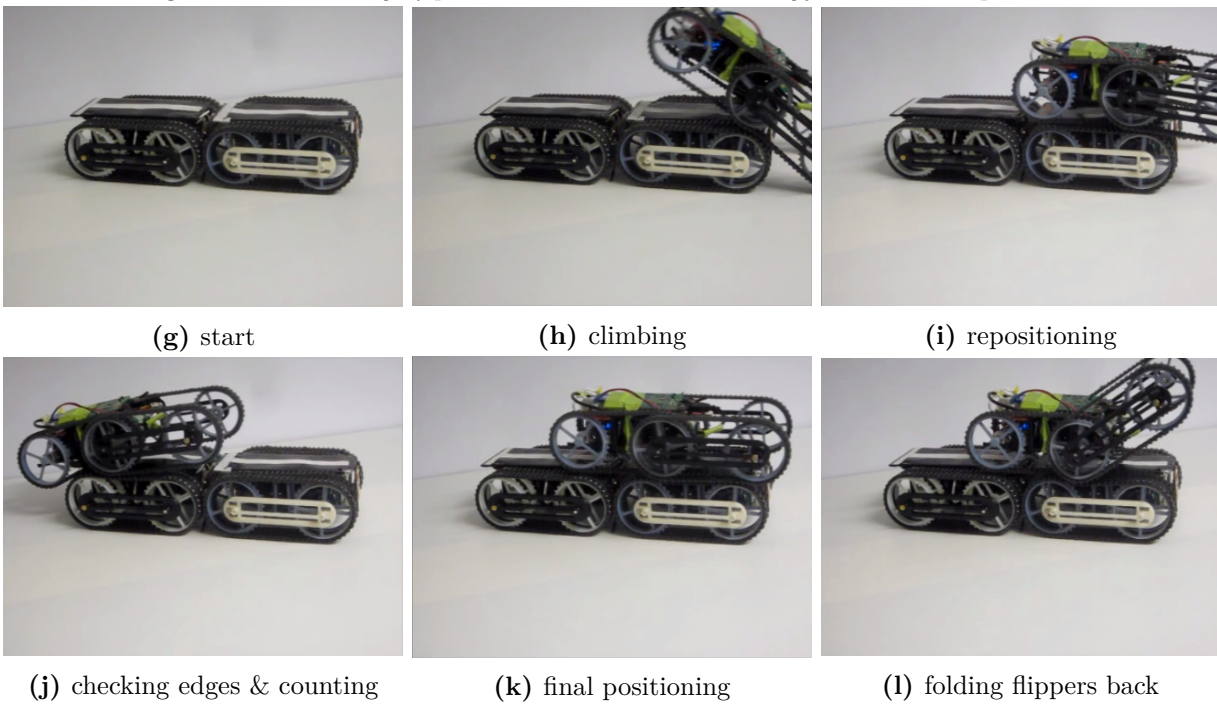


Figure 6.11: recording of phase III; the exact same strategy is reiterated on phase VI

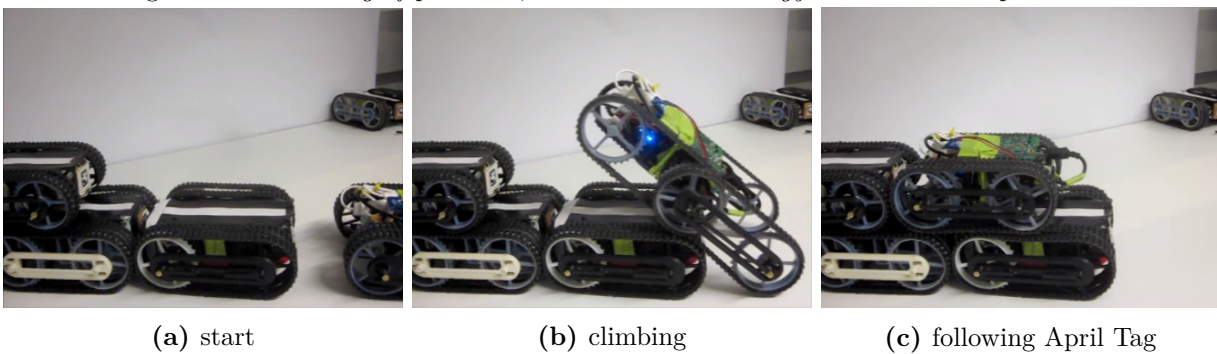


Figure 6.12: recording of phase V

As already mentioned in precedent chapters, April Tag orientation is used for navigation and alignment, IR sensors for navigation and counting when on top of another robot, accelerometer for detecting edge of the structure, and the encoders for small and precise maneuvers. The encoders are not connected to hardware interrupts pins of the Arduino, which means that when the encoders are used, no other sensors can be simultaneously read, and the robot is thus considered in open loop.

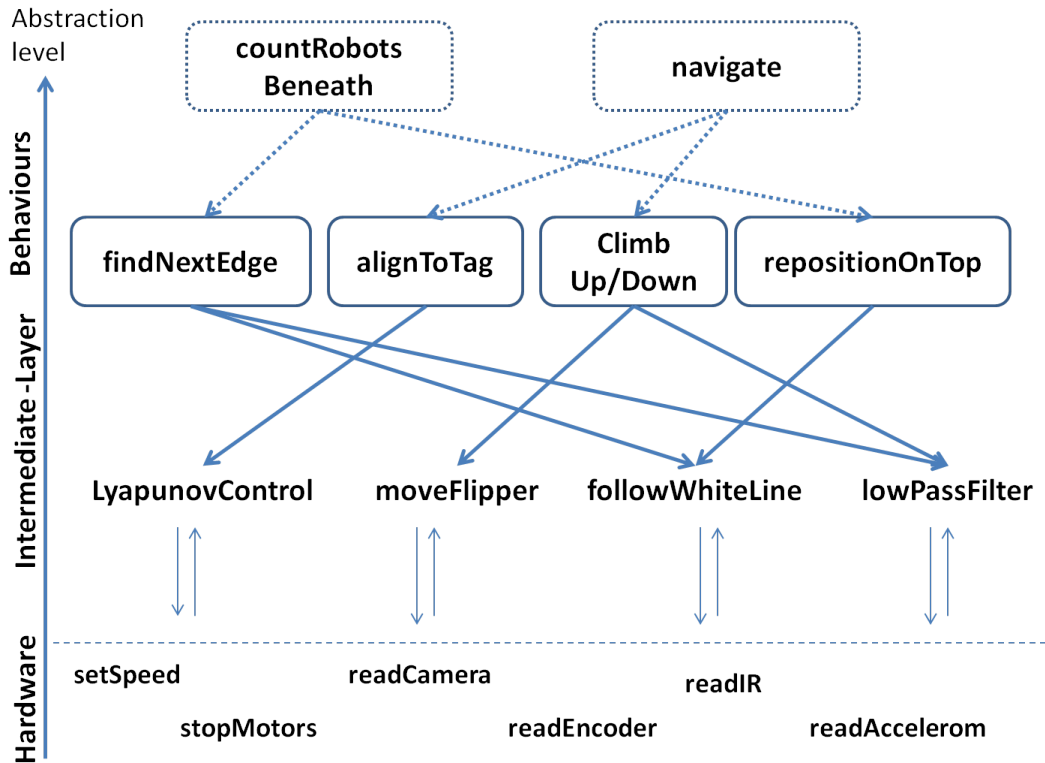


Figure 6.13: *top-to-bottom code implementation scheme, from high-level behaviors to hardware interfacing*

A more detailed pseudo-code for the instruction “count robots beneath” mentioned above is provided below with comments on the used sensors. The function is called as soon as the robot has stopped climbing (i.e. accelerometer indicates horizontal again) :

The key feature here is that the **robot uses its accelerometer to detect when it is about to fall**. IR sensors and motor encoders provide some additional information to increase the robustness of this sensing. However, this still remains one of the most common failures when running the algorithm.

In general, functions such as `countRobotsBeneath()` are decomposed in several behaviors, with the whole program being a finite-state machine implementation. Each behavior calls more intermediate-layer functions, which call other low-level, hardware-related functions or drivers, as shown in 6.13.

```
function COUNTROBOTSBENEATH()
```

```
    go back until the edge of the robot beneath is reached
```

```
    ▷ encoders, accelerometer
```

```
    while accelerometer indicates horizontal do
```

```
        ▷ accelerometer
```



```

    follow edge while counting the number of transverse white stripes           ▷ IR
  end while
  back up a little to avoid falling
  if one robot beneath counted then
    go down backwards, and realign with the April Tag; break;                 ▷ camera
  else if two robots beneath counted then
    follow line backwards until the middle of the two robots is reached       ▷ IR sensors
  end if
end function

```

The alignment using camera and April Tags with a non-holonomic robot is a non-trivial one. Not only the position, but also the orientation of the robot have to be simultaneously controlled. A closed-loop control law has to be found.

Closed-Loop Control

Given the non-holonomic system in figure 6.14, a control law is desired that links the orientation output by the camera to the wheel speed. By conducting a similar analysis as in [46], the following relationships are derived:

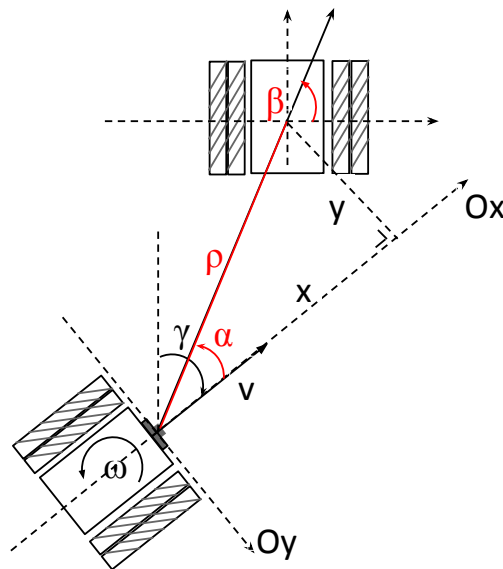


Figure 6.14: alignment with non-holonomic robots: ρ, α , and β have to be controlled to 0; x, y and γ are extracted using the camera, April Tags and homography

$$\rho = \sqrt{x^2 + y^2} \quad (6.1)$$

$$\alpha = -\text{atan}\left(\frac{y}{x}\right) \quad (6.2)$$

$$\beta = \alpha - \gamma - \frac{\pi}{2} \quad (6.3)$$

where x and y are the projections of the distance ρ in the local coordinate frame of the robot (Ox, Oy),

α its angle and γ the angle of the robot with respect to the stationary one (global frame). x, y and γ are given by the camera.

The desired control law should converge to

$$(\rho, \alpha, \beta) = (0, 0, 0)$$

. It can be shown that the control law:

$$v = k_\rho \cdot \rho \tag{6.4}$$

$$\omega = k_\alpha \cdot \alpha + k_\beta \cdot \beta \tag{6.5}$$

will yield a closed-loop system which is asymptotically stable if $k_\rho > 0$; $k_\beta > 0$; $k_\alpha - k_\rho > 0$, in accordance to **Lyapunov control theory** [47], [48]. It is also known that:

$$v = \frac{\dot{\phi}_1}{2} + \frac{\dot{\phi}_2}{2} \tag{6.6}$$

$$\omega = \frac{\dot{\phi}_1}{a} - \frac{\dot{\phi}_2}{a} \tag{6.7}$$

with v, ω the forwards, respectively rotational speed of the robot, $\dot{\phi}_i$ the motor angular speed, r the radius of the wheel, and a the axle length between wheels. Hence, by solving the above system, the following relationship linking motor speed orientation is derived

$$\dot{\phi}_1 = \frac{v}{r} - \frac{\omega a}{2r} = \frac{k_\rho \cdot \rho}{r} - \frac{k_\alpha \cdot \alpha + k_\beta \cdot \beta a}{2r} \tag{6.8}$$

$$\dot{\phi}_2 = \frac{v}{r} + \frac{\omega a}{2r} = \frac{k_\rho \cdot \rho}{r} + \frac{k_\alpha \cdot \alpha + k_\beta \cdot \beta a}{2r} \tag{6.9}$$

with $\dot{\phi}_1$ and $\dot{\phi}_2$ the left and right motor angular speed respectively

This relationship guarantees convergence if the orientation information ρ, α, β is available *at all times*. This is however not the case, as the camera has only a limited angle of vision of $\pm 30^\circ$, and when large maneuvers have to be done, the April Tag is easily lost.

6.3.2 Code

The main function implements 11 behaviors and calls 3 different classes. **Motor.cpp** implements all functions related to the motors and encoders, such as `setSpeed()`, `stop()`, `travelDistance()`, `useFlippers()`. **Sensors.cpp** regroups methods and other classes that read and are related to the sensors (accelerometer and IR sensors). **Control.cpp** implements the Lyapunov control with all related details. All the code is listed in the appendix.

The behaviour responsible for aligning to an April Tag (“followAprilTag”) combines both closed-loop

control and open-loop control. Open-loop control (relying only on encoders) is necessary when relatively large maneuvers are needed for alignment and the robot loses track of the April Tag. Closed-loop is mostly used to finish the alignment with precision.

Stopping climbing at the right moment plays a key role in the overall performance of the construction. As the robot does not know how many layers it has to climb, it is only relying on the accelerometer. The robot stops climbing when an inflexion point of the inclination angle is reached (derivative changes sign within a certain margin). After a short brake, and if it has not reached a horizontal position, the robot resumes its climbing . This is implemented in two behaviors (“goUp1”, “goUp2”).

As sensors may vary due to manufacturing reasons, and their placement may not be exactly the same at each robot assembly, an individual IR calibration of each robot is necessary. The minimum and the maximum value of each sensor when on black and white surface has to be replaced in the class constants. The output of the IR sensors is then linearized between these limits. Other than this, the same code can work without changes on all robots.

Chapter 7

Autonomy Assessment & Discussion

This chapter's goal is to have a quantitative idea of the performance of the self-assembly algorithm.

The 6 phases of the algorithm are separately assessed. As soon as the robot attains its goal, it is replaced by an empty one, the active robot is put back at the initial position, and the next assessment phase is conducted. For cost reasons, only 2 active robots have been built, the other ones have the same exterior shape but are empty inside. Additional weight is put to match the one of the active robot.

Each phase is assessed 7 times. Each trial is considered as a success only when the robot accomplishes the whole phase, and stops at the right position (within given tolerances). For phase II, IV and V, this means the alignment has to be within a 20cm radius from the ideal position, within a tolerance in the angle of $\pm 10^\circ$. This can be easily verified, as the alignment is done using the camera. For position III and VI, an error tolerance of 2 cm forwards and 1 cm laterally is accepted, as shown in 7.1

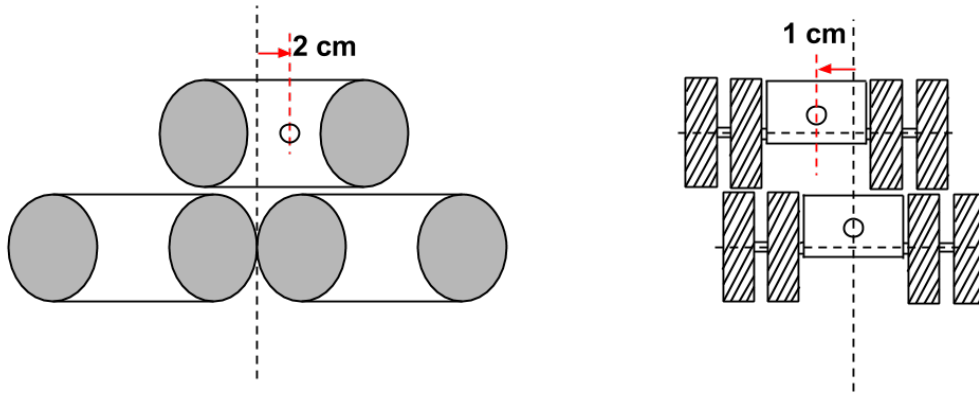


Figure 7.1: *accepted error tolerance for completion of position III and VI ; dimensions of the robot are 183 x 180 x 73 mm*

The following table lists the number of successes per phase, as well as the precision of the alignment in case of success for phase II, IV, and V. The min, max and median error value is given for x, y direction as well as for the angle. The error is computed relative to the ideal position, which is the one obtained

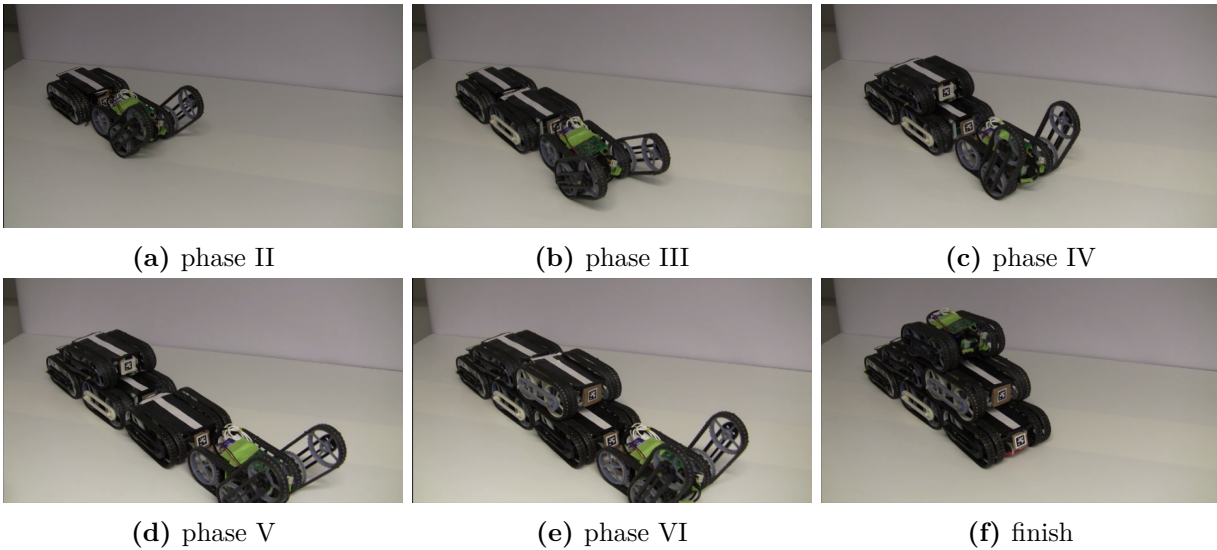


Figure 7.2: after the end of each assessment, an empty robot is placed at the same position the active one stopped previously

when manually putting the robot in the right place.

phase	successes	x (mm)			y(mm)			angle(°)		
		min	max	median	min	max	median	min	max	median
II	4/7	1	7	5.5	-2.5	1.5	-1.5	-6	-1.5	-4.5
III	7/7									
IV	4/7	-2	3	-0.7	-6	2.2	-0.2	-5	6	1.5
V	6/7	-3	4	1	-15.5	3	-11	-6.4	1.8	-2.4
VI	5/7									

Table 7.1: assessment of the 6 algorithm phases ; when available, the misalignment error relative to the ideal position is computed and the min, max and median values are listed

7.1 Discussion

When it comes to numbers of successful attempts, the robot performs the worst in phase II and IV. Failure reasons are mostly due to the difficulty of detecting in time the edge of the robot beneath, or mistaking while counting the number of passed robots.

The low-pass filter on the acceleration angle makes changes in angle occur with a certain delay, which reinforces the problem. The angle is computed based on the orientation of exterior forces exerted on the accelerometer. Therefore, less filtering would give too much importance to noise and spikes due to sudden accelerations. The IR sensors in the front of the robot are also used to detect when the robot comes over a gap. The sensors provide however with reliable results only within 2 cm, whereas when a robot is peaking over the edge this distance to the ground is around 9 cm , which makes the whole process unreliable.

In case of success, it can be noticed however that the error and its variance are negligible compared to the dimensions of the robot (180 x 183 mm). To give an idea of the scale, the results of phase II are

plotted at real scale in figure 7.3. The error variance for phase II and IV represents **less than 2 %** of its width. For these phases, a precise alignment within a **7mm** radius of the ideal position is achieved. The angle is always within $\pm 6^\circ$ from the ideal one.

In phase V, errors for the lateral displacement y are relatively higher, yet still negligible ($< 6.5\%$). The align algorithm is slightly different than in phase II and IV: in this case the robot does only line following until the forwards distance x to the next tag is within a certain boundary. While line-following, the displacement on y and the angle are neglected.

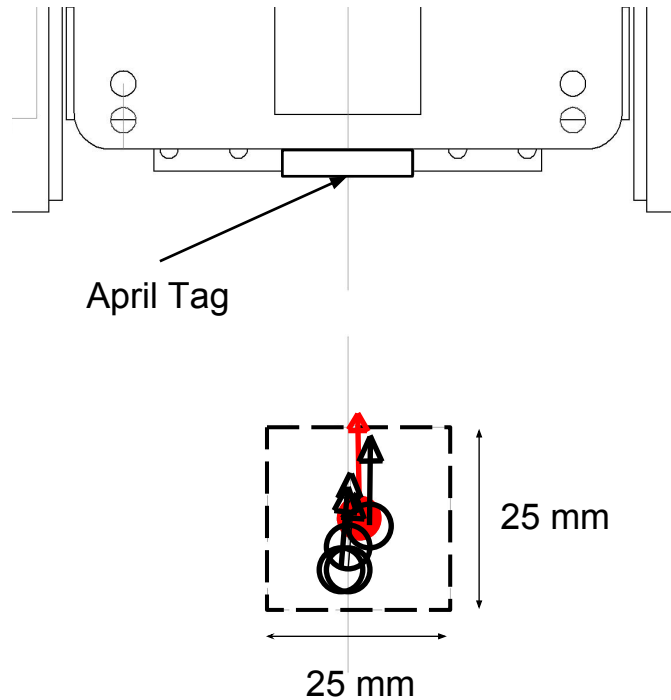


Figure 7.3: *alignment errors (for position and angle) at real scale for phase IV ; a 25mm square is drawn for illustrative purposes; the red mark is the ideal position (when the robot is aligned manually)*

From a hardware point of view, the combination between IR sensors, motor encodes and accelerometer should provide enough sensitivity to allow reliable detection. Therefore, this failures can be avoided with better edge detection functions, filtering, calibration or more accurate (IR) sensors.

By its design, the robot can be programmed such as to easily recover from a failure, return to the starting position and reiterate the process. Slots have been left on the sides of the robot for additional AprilTags, yet the algorithm implementation is left for future work.

From a cost perspective, the total of the mechanical and electrical parts rises to 297\$ per robot (including cables). Additionally, an estimated 60\$ is spent on 3D printing. The overall cost can be reduced by using a custom-designed PCB, custom made break-out boards for the sensors, and laser-cutting instead of 3D printing when possible.

Chapter 8

Conclusion & Future Work

A robot capable of robust climbing has been designed and programmed to form a 2D pyramidal structure. The algorithm has been designed in such a way as to allow formation of an arbitrary-sized structure. Except independent calibration, no changes have to be done when using the algorithm on another robot.

Separate assessment of each of the construction phases has shown that if the robot does not fall off the structure, and if it is possible to maneuver, alignment within $\pm 6^\circ$ and a 7 mm radius of the ideal position can be repeatably achieved.

“Climbot” is a novel design at the crossroads between modular and rough terrain exploration robots. Without docking or aligning mechanisms, it offers the possibility of exploring self-assembly in a less rigid, error-tolerant way. Moreover, by using its body as a controlled, smart and always available building block, it minimizes its dependency on the environment.

For demonstration purposes, a 2D pyramid has been chosen as goal structure, yet the climbing assessment of the robot shows that it is capable of climbing without getting stuck from all directions, thus allowing future work around more complicated structures and formations.

Although the software is designed to build a pyramid, its modularity allows for most of the lower-level function to be reused in other applications.

April Tags have been used for navigation and alignment, yet the on-board hardware and resources allow for other image processing methods to be exploited.

All in all, this work has provided the optimization methods, mechanics, electronics and computational power to create a research platform for collective structure building through self-assembly.

8.1 Future Work

The most obvious next step to do would be to implement a failure recovery algorithm, allowing the robot to return to the right spot and to re-attempt climbing after falling.

The 2D pyramid demonstration does not exploit the full mechanical potential of the robot. More complex shapes, like 3D pyramid, ramps or structures using environment obstacles could be considered in future work. If desired, an alignment/docking, or even a *transportation* system (à la Kiva system !) can comfortably be built on a deck.

More advanced planning, navigation and feature detection can always be implemented using the Raspberry PI and its camera.

On the electrical perspective, a redesign of the electronics with a custom PCB would make the cost of a single robot drop and facilitate assembly, thus allowing for the construction of a larger team. On the mechanical perspective, a more rugged design would allow for better resistance to falls. Compliant wheels or elastic elements could be used to damp such shocks.

Exploring issues when several robots are running concurrently is also an essential part of collective robotic behaviour. Decision making over which robot to go first, where and when represents a interesting challenge.

8.2 Acknowledgements

For helping me completing this work, I wish to thank Mike Rubenstein for his advice and tips. Special thanks go to Kirstin Petersen, for the wonderful discussions and feedback around mechanics, and to Alex Cornejo for sharing his knowledge on embedded systems, programming ... and Linux. I thank Michael S. Kester for Saturday debugging and late-night chats.

Nothing can be achieved without friendship, so I also thank Sebastian Gabor, Marius Alexandrescu, Tristan Vouga, David Formica and all my friends from Lausanne for reminding me of that despite the distance.

Finally, I thank prof. Radhika Nagpal for her cheerful advising and prof. D. Floreano for enabling me to do this exchange.

Bibliography

- [1] Nathan J. Mlot, Craig A. Tovey, and David L. Hu. Fire ants self-assemble into waterproof rafts to survive floods. *Proceedings of the National Academy of Sciences*, 2011.
- [2] C. Anderson, G. Theraulaz, and J.-L. Deneubourg. Self-assemblages in insect societies. *Insectes Sociaux*, 49(2):99–110, 2002.
- [3] <http://www.dailymail.co.uk/news/article-2603204/No-bridge-far-Army-ants-stranded-tree-build.html>. Accessed July, 2014.
- [4] K. Gilpin and D. Rus. Modular robot systems. *Robotics Automation Magazine, IEEE*, 17(3):38–55, Sept 2010.
- [5] K. Gilpin, K. Kotay, and D. Rus. Miche: Modular shape formation by self-dissassembly. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2241–2247, April 2007.
- [6] K. Gilpin, K. Koyanagi, and D. Rus. Making self-disassembling objects with multiple components in the robot pebbles system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3614–3621, May 2011.
- [7] Byoung Kwon An. Em-cube: cube-shaped, self-reconfigurable robots sliding on structure surfaces. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3149–3155, May 2008.
- [8] J.W. Romanishin, K. Gilpin, and D. Rus. M-blocks: Momentum-driven, magnetic modular robots. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4288–4295, Nov 2013.
- [9] Y. Suzuki, N. Inou, M. Koseki, and H. Kimura. Reconfigurable group robots adaptively transforming a mechanical structure - extended criteria for load-adaptive transformations -. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 877–882, Sept 2008.
- [10] J. Davey, Ngai Kwok, and M. Yim. Emulating self-reconfigurable robots - design of the smores system. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4464–4469, Oct 2012.
- [11] B. Salemi, M. Moll, and Wei-Min Shen. Superbot: A deployable, multi-functional, and modular self-reconfigurable robotic system. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3636–3641, Oct 2006.

- [12] M. Yim, B. Shirmohammadi, J. Sastra, M. Park, M. Dugan, and C.J. Taylor. Towards robotic self-reassembly after explosion. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2767–2772, Oct 2007.
- [13] Roland Siegwart, Pierre Lamon, Thomas Estier, Michel Lauria, and Ralph Piguet. Innovative design for wheeled locomotion in rough terrain. *Robotics and Autonomous Systems*, 40:151 – 162, 2002. Intelligent Autonomous Systems - {IAS} -6.
- [14] Yugang Liu and Guangjun Liu. Track–stair interaction analysis and online tipover prediction for a self-reconfigurable tracked mobile robot climbing stairs. *Mechatronics, IEEE/ASME Transactions on*, 14(5):528–538, Oct 2009.
- [15] Bin Li, Shugen Ma, Bin Li, Minghui Wang, and Yuechao Wang. A dynamic shape-shifting method for a transformable tracked robot. In *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, pages 466–471, Dec 2010.
- [16] Jinguo Liu, Yuechao Wang, Shugen Ma, and Bin Li. Analysis of stairs-climbing ability for a tracked reconfigurable modular robot. In *Safety, Security and Rescue Robotics, Workshop, 2005 IEEE International*, pages 36–41, June 2005.
- [17] H. Benjamin Brown, J.M. Vande Weghe, C.A Bererton, and P.K. Khosla. Millibot trains for enhanced mobility. *Mechatronics, IEEE/ASME Transactions on*, 7(4):452–461, Dec 2002.
- [18] Zhiqing Li, Shugen Ma, Bin Li, Minghui Wang, and Yuechao Wang. Design and basic experiments of a transformable wheel-track robot with self-adaptive mobile mechanism. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1334–1339, Oct 2010.
- [19] M. Freese, M. Kaelin, J.-M. Lehky, G. Caprari, T. Estier, and R. Siegwart. Lamalice: a nanorover for planetary exploration. In *Micromechatronics and Human Science, 1999. MHS '99. Proceedings of 1999 International Symposium on*, pages 129–133, 1999.
- [20] Xingguang Duan, Qiang Huang, Nasir Rahman, Junchen Li, and Jingtao Li. Mobit, a small wheel - track - leg mobile robot. In *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, volume 2, pages 9159–9163, 2006.
- [21] <http://www.icm.cc/>. Accessed July, 2014.
- [22] Yili Fu, Zhihai Li, Hejin Yang, and Shuguo Wang. Development of a wall climbing robot with wheel-leg hybrid locomotion mechanism. In *Robotics and Biomimetics, 2007. ROBIO 2007. IEEE International Conference on*, pages 1876–1881, Dec 2007.
- [23] L. Briones, P. Bustamante, and M.A Serna. Wall-climbing robot for inspection in nuclear power plants. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 1409–1414 vol.2, May 1994.
- [24] T. Miyake, H. Ishihara, and T. Tomino. Vacuum-based wet adhesion system for wall climbing robots -lubricating action and seal action by the liquid-. In *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*, pages 1824–1829, Feb 2009.

- [25] Che-Seung Cho, Jin-Dae Kim, Sung-Gun Lee, Sung Kyu Lee, Seung-Chul Han, and Byeong-Soo Kim. A study on automated mobile painting robot with permanent magnet wheels for outer plate of ship. In *Robotics (ISR), 2013 44th International Symposium on*, pages 1–4, Oct 2013.
- [26] M. Wagner, Xiaoqi Chen, M. Nayerloo, Wenhui Wang, and J.G. Chase. A novel wall climbing robot based on bernoulli effect. In *Mechronic and Embedded Systems and Applications, 2008. MESA 2008. IEEE/ASME International Conference on*, pages 210–215, Oct 2008.
- [27] F. et al. Rochat. Cy-mag3d: magnetic climbing inspection robot. *Climbing and Walking Robots (CLAWAR)*, 2011.
- [28] F. et al. Rochat.
- [29] Chang-Hwan Choi, Seung-Ho Jung, and Seung-Ho Kim. Feeder pipe inspection robot using an inch-worm mechanism with pneumatic actuators. In *Robotics and Biomimetics, 2004. ROBIO 2004. IEEE International Conference on*, pages 889–894, Aug 2004.
- [30] F. Nigl, Shuguang Li, J.E. Blum, and H. Lipson. Structure-reconfiguring robots: Autonomous truss reconfiguration and manipulation. *Robotics Automation Magazine, IEEE*, 20(3):60–71, Sept 2013.
- [31] R. Saltaren, R. Aracil, O. Reinoso, and M.A Scarano. Climbing parallel robot: a computational and experimental study of its performance around structural nodes. *Robotics, IEEE Transactions on*, 21(6):1056–1066, Dec 2005.
- [32] M. Abderrahim, C. Balaguer, A Gimenez, J. M. Pastor, and V.M. Padron. Roma: a climbing robot for inspection operations. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 3, pages 2303–2308 vol.3, 1999.
- [33] M. Tavakoli, A Marjovi, L. Marques, and AT. de Almeida. 3dclimber: A climbing robot for inspection of 3d human made structures. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 4130–4135, Sept 2008.
- [34] Tin Lun Lam and Yangsheng Xu. A flexible tree climbing robot: Treebot - design and implementation. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5849–5854, May 2011.
- [35] Mondada Francesco Bonani Michael Siegwart Roland Yves Scheidegger, Noemy. Bi-pedal robot for rescue operations. *International conference on climbing and walking robot (CLAWAR)*, 2006.
- [36] H. Prahlad, R. Pelrine, S. Stanford, J. Marlow, and R. Kornbluh. Electroadhesive robots x2014;wall climbing robots enabled by a novel, robust, and electrically controllable adhesion technology. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3028–3033, May 2008.
- [37] Y. Yoshida and Shugen Ma. Design of a wall-climbing robot with passive suction cups. In *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, pages 1513–1518, Dec 2010.

- [38] Sangbae Kim, A.T. Asbeck, M.R. Cutkosky, and W.R. Provancher. Spinybotii: climbing hard walls with compliant microspines. In *Advanced Robotics, 2005. ICAR '05. Proceedings., 12th International Conference on*, pages 601–606, July 2005.
- [39]
- [40] Ozgur Unver, A. Uneri, A. Aydemir, and M. Sitti. Geckobot: a gecko inspired climbing robot using elastomer adhesives. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2329–2335, May 2006.
- [41] E. Z. Moore, D. Campbell, F. Grimminger, and M. Buehler. Reliable stair climbing in the simple hexapod 'rhex'. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 3, pages 2222–2227, 2002.
- [42] Justin Werfel, Kirstin Petersen, and Radhika Nagpal. Designing collective behavior in a termite-inspired robot construction team. *Science*, 343(6172):754–758, 2014.
- [43] <http://www.pololu.com/>. Accessed July, 2014.
- [44] Guangming Song, Hui Wang, Jun Zhang, and Tianhua Meng. Automatic docking system for recharging home surveillance robots. *Consumer Electronics, IEEE Transactions on*, 57(2):428–435, May 2011.
- [45] E. Olson. Apriltag: A robust and flexible visual fiducial system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3400–3407, May 2011.
- [46] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA, 2004.
- [47] A Astolfi. Exponential stabilization of a car-like vehicle. In *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, volume 2, pages 1391–1396 vol.2, May 1995.
- [48] A Tayebi and A Rachid. Discontinuous control for exponential stabilization of wheeled mobile robots. In *Intelligent Robots and Systems '96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on*, volume 1, pages 60–65 vol.1, Nov 1996.

Appendix A

Code Details

Listing A.1: *ClimBot.ino*

```
#include <stdio.h>
#include <string.h>
#include <SoftwareSerial.h>
#include <QTRSensors.h>
#include <Wire.h>
#include <LSM303.h>

#include "Sensors.h"
#include "Control.h"
#include "Motors.h"

bool DEBUG = 1;
bool FILTER =0;
bool CLIMB =1, STAY=0;

const int pinTx = 7, pinRx =8;
const int fwds =1, bwds = -1;

const int baseLength = 3;

const int cw = 1, ccw = -1;
const int accelX = A4, accelY = A5, accelZ = 11;
//const int numIrSensors = 4; // it's a global variable so Serial.h sees it
const int speed = 220;
const int speedBack = 220;
const int climbSpeed = 190;
const int followLineSpeed = 240; //robot #1
//const int followLineSpeed = 230; //robot #2
const int robotLength = 1400; //in nb of encoder tours

enum behaviour{search, AprilTag, pos, peak, peakBack, goDown, adjust, stop,
               goUp,goUpInter, alignAprilGround, alignAprilTop};
behaviour myBehaviour = AprilTag;
```

```

//max camera sight
// -24, 45 recorder for a_max b_max, however
float x_max = 235, y_max = 39, a_max = -10/180*PI, b_max = 44/180*PI;

// conditions for convergent control
const int minRhoDeploy = 73;
const int minRhoAlign = 53;
const int rhoPerim = 260; // at 300 roll is not valid anymore

const int rollThresh = 9;

const int window = 5;
const int thetaHor = 6; // calibrate by putting robot on another one
const int thetaTol = 4; // tolerance on horizontality

typedef struct position{
    float x;
    float y;
    float z;
    float roll;
    float pitch;
    float yaw;
} position;
MotorsClass myMotors;
ControlClass myControl;
LSM303 myLSM;
SensorsClass IRsensors;

position myPos;

char inData[100];
int motorSpeed[2];

//int flag =1; //lyap controller
bool flag =0; //line following
bool lostTrace =0;
bool oLoop =1;

int climbCount = 0;
int posCount= 0;
int turnCount = 0;
float rollPrev = 90;

//accelerom
float theta=0, thetaPrev = 0;
float thetaFilt>window ={}; //C++ init to 0
double accelMag =0;
int i=0; //lousy filter counter

```

```

int sensors[numIrSensors];
int sLeft, sRight;
float rho;

SoftwareSerial mySerial(pinRx, pinTx); // RX, TX

inline int Sign(float f) {
    return ((f>0) - (f<0)) ;
}

float getDistance() {
    float x = sqrt(myPos.x*myPos.x + myPos.y*myPos.y);
    return x;
}

bool getSerial()
{
    int bufferSize;
    char junk;

    if ((bufferSize =Serial.available())==sizeof(position))
    {
        for (int j=0; j<sizeof(position);j++)
            inData[j] = Serial.read();

        myPos.x = (*(position*)inData).x;
        myPos.y = (*(position*)inData).y;
        myPos.z = (*(position*)inData).z;
        myPos.roll = (*(position*)inData).roll;
        myPos.pitch = (*(position*)inData).pitch;
        myPos.yaw = (*(position*)inData).yaw;

        return 1;
    }

    else if (bufferSize > sizeof(position))
        //just empty buffer and ignore the data
        while(Serial.available())
            junk = Serial.read();

    return 0;
}

//follows IR sensors until loss of balance or stripe is reached
void followLine(int direction)
{
    if (theta<-4 || theta >20)
    {
        myMotors.stop_motors();
        //mySerial.println("stopped bc of theta");
    }

    else if(sensors[0] < lineRange/2 ||sensors[2] < lineRange/2)

```

```

{
    //if(sensors[0] > 0 & sensors[2] > 0)
        //myMotors.set_speed(230- sensors[2], 230- sensors[0]);
        myMotors.set_speed(Sign(direction)*(followLineSpeed- (
            lineRange- sensors[2])),
                                Sign(direction)*(
                                    followLineSpeed -
                                        (lineRange -
                                            sensors[0])));

        //mySerial.println("following line right now");
    }
    // forwards if all B
    else
        myMotors.set_speed(Sign(direction)*speed, Sign(direction)*speed);
}

void followLineBwds()
{
    int x,y;
    if (theta<0 || theta >20)
        myMotors.stop_motors();

    else if (sensors[3] > lineRange/2 ||sensors[4] > lineRange/2)
    {

        x = - 235 + sensors[4];
        y = - 235 + sensors[3];
        myMotors.set_speed(x, y);
        //mySerial.println(x); mySerial.println(y);
    }

    else
        myMotors.set_speed(-speed, -speed);
}

void posFct()
{
    // if I see an AprilTag I go for it
    if(getSerial())
    {
        myMotors.stop_motors();
        delay(100);
        myMotors.retractFlip();
        delay(300);
        myBehaviour= alignAprilTop;
    }

    else if(sensors[0] < whiteThresh & sensors[1] < whiteThresh & sensors[2] <
        whiteThresh)
        ///| sensors[0] > blackThresh & sensors[1] > blackThresh &

```



```

        sensors[2] > blackThresh) // not sure about this
    {
        myBehaviour = peak;
        // be sure that you go past the white stripe
        //mySerial.println("transition");
        myMotors.moveEnc(fwds,70,speed); // instead of 180
        myMotors.stop_motors();
        delay(100);
        posCount++;
    }
    else
        //go to next white line
        followLine(fwds);
}

void peakFct()
{
    //comment
    //go forward until accelerom changes or another WHITE spot is detected
    if (theta < thetaHor - 7 )
    {
        //I am on a cliff
        myBehaviour = peakBack;
        myMotors.moveEnc(bwds,800,200);
        //delay(1500);
        myMotors.stop_motors();
    }
    else if (sensors[0] < whiteThresh || sensors[1] < whiteThresh || sensors[2] <
        whiteThresh & (theta > thetaHor))
        //I am on another robot
        myBehaviour =pos;
    else if(getSerial())
    {
        myMotors.stop_motors();
        delay(100);
        myMotors.retractFlip();
        delay(300);
        myBehaviour= alignAprilTop;
    }
    else
    {
        //mySerial.println("trying to open loop forward");
        myMotors.set_speed(speed,speed);
    }
}

void peakBackFct()
{
    //if only one robot under me, go down, else adjust in the middle of the 2

```

```

    precedents ones
  if(posCount ==1)
  {
    myBehaviour = goDown;
    myMotors.moveEnc(bwds,robotLength*2,speedBack);
    myMotors.stop_motors();
    delay(600);
  }
  else if (posCount>1 & theta < 10 & theta > 3)
  {

    //myMotors.moveEnc(bwds,800,speed);
    //myMotors.stop_motors();
    myBehaviour = adjust;

    //mySerial.println("switched to adjust");
  }
  else
  {
    myMotors.stop_motors();
    mySerial.println("I am in peakBack mode without having counted any
      white stripe");
  }

  if(sensors[0] < whiteThresh & sensors[1] < whiteThresh & sensors[2] <
    whiteThresh)
  {
    //you skipped this line but it's not too late!
    myMotors.moveEnc(fwds,300);
    myBehaviour =adjust;
  }
}

void goUpFct()
{
  //if there is too large change in angle, wait
  if (thetaPrev- theta > 4) //instead of theta < 10
  {
    myMotors.stop_motors();
    //allow enough time for accelerometer to calm down (filter !)
    delay(500);
    myBehaviour= goUpInter;
  }
  else
    myMotors.moveEnc(fwds,90,175); //need some serious momentum here //
      instead of 175
}

void goUpInterFct()
{

```

```

//after detecting a chaning in the angle, now I wait to see if gravity
//pulls me back flat
if(theta >thetaHor + 4 )
{
    delay(300);
    climbCount++;
    if (climbCount==7)
        myBehaviour = goUp;
}
else
{
    myBehaviour=pos;
    //myMotors.retractFlip();
    delay(1500);
    myMotors.moveEnc(bwds,180,225); // with flippers retracted was 400 !
    delay(1000);
}
}

void goDownFct()
{
    if(theta > 10)
    {
        //if still inclined, go down
        myMotors.moveEnc(bwds, 250 , speedBack);
        //followLineBwds();
    }
    else
    {
        //just go back enough to see the AprilTag
        myMotors.moveEnc(bwds,robotLength/3,speed);
        myMotors.stop_motors();
        //followLineBwds();
        //TOBE replaced !
        myMotors.retractFlip();
        delay(1500);
        myBehaviour = alignAprilGround;
        posCount=0;
    }
}

void adjustFct()
{
    if (theta> 10)
    {
        // position counting is wrong
        posCount=1;
        myBehaviour = goDown;
    }
    else if( sensors[0] > lineRange/2 & sensors[1] > lineRange/2 & sensors[2] >
        lineRange/2 & theta < 10 & theta > 0)

```

```

    {
        //adjustment manoeuver;
        //myMotors.moveEnc(fwds,150);
        myMotors.stop_motors();
        //mySerial.println("I am adjusted");
        myMotors.retractFlip();
        delay(1500);
        myBehaviour=stop;
    }
    else
        followLineBwds();
}

void searchFct()
{
    if(flag)
    {
        myMotors.set_speed(-190,-190, 1000);
        myBehaviour = AprilTag;
    }
    else
    {
        if(turnCount < 8)
        {
            myMotors.turn(cw, speed, 300);
            turnCount++;
        }
        else
        {
            myMotors.turn(ccw, speed, 1800);
            turnCount=0;
            myMotors.set_speed(190,190,1000);
        }
    }
    myMotors.stop_motors();
    delay(600);
}

void alignAprilTopFct()
{
    int flag= getSerial();
    float rho = getDistance();

    if(flag & myPos.x != 0.0 & myPos.y != 0.0 & myPos.z != 0.0
    & myPos.roll!= 0.0 & myPos.pitch!= 0.0 & myPos.yaw!= 0.0)
    {
        if(rho > minRhoDeploy)
            followLine(fwds);
            //myMotors.retractFlip();
        else

```

```

        myBehaviour= stop;
    }
}

void followAprilTag(bool climb)
{
    //acquire data
    flag = getSerial();

    myControl.lyap(myPos.x, myPos.y, myPos.roll, motorSpeed);
    myControl.normalize(motorSpeed);

    sLeft = motorSpeed[0];
    sRight = motorSpeed[1];

    rho=getDistance();

    //there has been a detection and this detection is valid
    if ( flag & myPos.x != 0.0 & myPos.y != 0.0 & myPos.z != 0.0
    & myPos.roll!= 0.0 & myPos.pitch!= 0.0 & myPos.yaw!= 0.0)
    {
        if ( rho < minRhoDeploy & abs(myPos.roll) < rollThresh & climb)
        {
            myMotors.deployFlip();

            myMotors.set_speed(160,160,1200);
            myBehaviour= goUp;

            myMotors.stop_motors();
            delay(200); // wait a sec for IR
        }
        else if ( rho < minRhoAlign & abs(myPos.roll) < rollThresh & !climb)
            myBehaviour= stop;

        else if (rho < minRhoDeploy +10 & abs(myPos.roll) > rollThresh)
        {
            //if too close and roll to big, maneuver to reduce it
            //using encoders
            myMotors.moveEnc(bwds,1000);
            myMotors.turnEnc(-Sign(myPos.roll),400,190);
            myMotors.moveEnc(fwds,600);
            myMotors.turnEnc(Sign(myPos.roll),600,190);
        }
        else if (abs(rollPrev)+2 < abs(myPos.roll) & myPos.roll> 30 & oLoop)
        {
            //if change in roll angle too sudden, trajectory needs to
            overshoot,
            //camera will be lost -> openLoop

            myMotors.turnEnc(ccw,400,190);
        }
    }
}

```

```

        myMotors.moveEnc(fwds,600);
        myMotors.stop_motors();
        myMotors.turnEnc(cw,600,190);
        oLoop =0;
    }
    else
    {
        //follow Lyapunov
        myMotors.set_speed(sLeft, sRight,50);
        oLoop =1;
    }
    rollPrev = myPos.roll;
}
myMotors.stop_motors();
//delay important, else getSerial jams (is this really the reason?)
delay(100);
}

void setup()
{
    Serial.begin(9600);
    //caution with intereferances between Hard- and Soft- Serials
    mySerial.begin(9600); //using PGM03A for debugging
    myMotors.init();
    Wire.begin();
    myLSM.init();
    myLSM.enableDefault();
}

void loop()
{
    IRsensors.readRaw();
    IRsensors.read(sensors, FILTER,DEBUG);
    myLSM.read();

    //TODO: fix the issue with the accelerom angle
    accelMag = sqrt(((float)myLSM.a.x/1000)*((float)myLSM.a.x/1000) +
                    ((float)myLSM.a.y/1000)*((float)myLSM.a.y
                    /1000)+
                    ((float)myLSM.a.z/1000)*((float)myLSM.a.z
                    /1000));

    if (myLSM.a.x < 0)
        theta = - (180 - acos((float)myLSM.a.z/1000/accelMag)*180/PI);
    else
        theta = 180- acos((float)myLSM.a.z/1000/accelMag)*180/PI ;

    //filter acceleration angle
    thetaFilt[i]= theta;

```

```
i++;
if (i==window)
    i=0;
theta = 0; // use same variable for filter result
for(int j=0;j <window;j++)
    theta += thetaFilt[j];
theta /= window;

if(DEBUG)
{
    mySerial.print(" theta: "); mySerial.println(theta);
}

switch (myBehaviour)
{
    case search:
        searchFct();
        break;

    case AprilTag:
        followAprilTag(CLIMB);
        break;

    case goUp:
        goUpFct();
        break;

    case goUpInter:
        goUpInterFct();
        break;

    case pos:
        posFct();
        break;

    case peak:
        peakFct();
        break;

    case peakBack:
        peakBackFct();
        break;

    case adjust:
        adjustFct();
        break;

    case goDown:
        goDownFct();
        break;
```

```
    case alignAprilGround:
        //if not flat yet, go flat first
        if (theta > thetaHor +4)
            myMotors.moveEnc(bwds, 30);
        else
            followAprilTag(STAY);

        break;

    case alignAprilTop :
        alignAprilTopFct();
        break;

    case stop:
        myMotors.stop_motors();

    default:

        break;
}
flag =0;
thetaPrev =theta;

//if robot is lifted = reset pos
if (theta < -30)
{
    myBehaviour =AprilTag;
    posCount=0;
}
if(DEBUG){
    mySerial.print(myBehaviour);
    mySerial.print(" with a positionCount ");
    mySerial.println(posCount);
    mySerial.println();
}
}
```


Listing A.2: *Motors.h*

```

// Motors.h

/*
298:1 Pololu Micro Motors, with optical encoders on extended shaft (resolution: 6)
double threaded worm gear with 30 teeth = > 4470:1 ratio
should take 11175 motor turns to turn flippers 180deg (theoretically!)
*/

#ifndef _MOTORS_h
#define _MOTORS_h

#if defined(ARDUINO) && ARDUINO >= 100
    #include "Arduino.h"
#else
    #include "WProgram.h"
#endif

//proper way to do it : use private variables and use initializer list
//pins should come as arguments of the init() function

const int r_motor_n = 13; //PWM control Right Motor -
const int r_motor_p = 9; //PWM control Right Motor +
const int l_motor_p = 10; //PWM control Left Motor +
const int l_motor_n = 11; //PWM control Left Motor -

const int arm_motor_p = 6; // PWM flipper +
const int arm_motor_n = 5; // PWM flipper -

const int encPinFlip = A5;
const int encPinRear = A4;
//motor encoder limit values : vary only slightly from one motor to another
const int encoderMax = 845;
const int encoderMin = 47;

const int nbToursFlip = 13400; // corresponding to 180deg turn of flippers

const int maxMotorSpeed = 255; // in fact corresponds to minimum
const int Impetus = 50; // delay in ms //behaviour with 10 ms

const int flipperSpeed = 100;

class MotorsClass
{
protected:
    int leftSpeed;
    int rightSpeed;

```

```
    unsigned long encoder;

    int comp;

    int limitSpeed(int); // protects from overflow variables
    int readEncoder(const int); //simple ADC conversion
    void turnUntil(const long, const int);

public:
    void init(); // this is not a constructor
    void set_speed(int , int , int dt=10);
    void turn(int, int speed = 200, int dt=100);
    void deployFlip();
    void retractFlip();
    void stop_motors();

    //move fuds/bwds and turn cw/ccw using encoder (on one wheel only)
    void moveEnc(int dir, long tours, int speed =200);
    void turnEnc(int dir, long tours, int speed =200);

};

//extern MotorsClass Motors;

#endif
```

Listing A.3: *Motors.cpp*

```

//
//
//
#include "Motors.h"

void MotorsClass::init()
{
    pinMode(r_motor_n, OUTPUT); //Set control pins to be outputs
    pinMode(r_motor_p, OUTPUT);
    pinMode(l_motor_p, OUTPUT);
    pinMode(l_motor_n, OUTPUT);
    digitalWrite(r_motor_n, LOW); //set both motors off for start-up
    digitalWrite(r_motor_p, LOW);
    digitalWrite(l_motor_p, LOW);
    digitalWrite(l_motor_n, LOW);
}

int MotorsClass::limitSpeed(int x)
{
    if (x > maxMotorSpeed)
        return maxMotorSpeed;
    if (x < -maxMotorSpeed)
        return -maxMotorSpeed;

    return x;
}

void MotorsClass::set_speed(int left, int right, int dt )
{
    rightSpeed = limitSpeed(right);
    leftSpeed = limitSpeed(left);

    if( rightSpeed<0 )
    {
        digitalWrite(r_motor_p, HIGH); //Set motor direction, 1 low, 2 high
        analogWrite(r_motor_n, abs(rightSpeed));
    }
    else if (rightSpeed >0)
    {
        digitalWrite(r_motor_n, HIGH); //Set motor direction, 1 low, 2 high
        analogWrite(r_motor_p, rightSpeed);
    }
    else
    {
        digitalWrite(r_motor_p, LOW); //Set motor direction, 1 low, 2 high
        digitalWrite(r_motor_n, LOW);
    }
}

```

```
    }

    if(leftSpeed < 0)
    {
        digitalWrite(l_motor_p, HIGH); //Set motor direction, 1 low, 2 high
        analogWrite(l_motor_n, abs(leftSpeed));
    }
    else if (leftSpeed >0)
    {
        digitalWrite(l_motor_n, HIGH); //Set motor direction, 1 low, 2 high
        analogWrite(l_motor_p, leftSpeed);
    }
    else
    {
        digitalWrite(l_motor_n, LOW); //Set motor direction, 1 low, 2 high
        digitalWrite(l_motor_p, LOW);
    }

    delay(dt);
}

void MotorsClass::stop_motors()
{
    digitalWrite(r_motor_n, LOW); //Set motor direction, 1 low, 2 high
    digitalWrite(r_motor_p, LOW);
    digitalWrite(l_motor_p, LOW);
    digitalWrite(l_motor_n, LOW);
    digitalWrite(arm_motor_p, LOW);
    digitalWrite(arm_motor_n, LOW);
    leftSpeed=0;
    rightSpeed=0;
}

void MotorsClass::deployFlip()
{
    digitalWrite(arm_motor_n, HIGH); //Set motor direction, 1 low, 2 high
    analogWrite(arm_motor_p, flipperSpeed);

    turnUntil(nbToursFlip, encPinFlip);
    stop_motors();
}

void MotorsClass::retractFlip()
{
    digitalWrite(arm_motor_p, HIGH); //Set motor direction, 1 low, 2 high
    analogWrite(arm_motor_n, flipperSpeed);

    turnUntil(nbToursFlip, encPinFlip);
    stop_motors();
}
```

```

int MotorsClass::readEncoder(const int encPin)
{
    int x = analogRead(encPin);

    if (x > (encoderMin+encoderMax)/2)
        return 1 ;
    else
        return 0;
}

void MotorsClass::turnUntil(const long nbTurns , const int pin)
{
    //measure every change of of ADC
    encoder =0;
    int x, xp=0;

    while(encoder <nbTurns)
    {
        x = readEncoder(pin);
        if (x != xp)
            encoder++;

        xp = x;
    }
}

void MotorsClass::turn(int direction , int speed , int dt)
{
    //turn CW
    if (direction > 0)
        set_speed(speed , -speed , dt);
    //CCW
    else
        set_speed(-speed , speed , dt);
}

void MotorsClass::moveEnc(int direction , long nbTours , int speed)
{
    //forwards
    if(direction > 0)
    {
        set_speed(speed , speed);
        turnUntil(nbTours , A4);
    }
    else
    {
        set_speed(-speed , -speed);
        turnUntil(nbTours , encPinRear);
    }
}

```

```
    }  
}  
  
void MotorsClass::turnEnc(int direction, long nbTours, int speed)  
{  
    //cw  
    if(direction > 0)  
    {  
        set_speed(speed, -speed);  
        turnUntil(nbTours, A4);  
    }  
    //ccw  
    else  
    {  
        set_speed(-speed, speed);  
        turnUntil(nbTours, encPinRear);  
    }  
}
```

Listing A.4: *Sensors.h*

```

// Sensors.h

#ifndef _SENSORS_h
#define _SENSORS_h

#if defined(ARDUINO) && ARDUINO >= 100
    #include "Arduino.h"
#else
    #include "WProgram.h"
#endif
#include <SoftwareSerial.h>

//IR sensor constants
const int IR1 = A0, IR2 = A1, IR3 = A2, IR4 = A11, IR5= A6;
const int numIrSensors = 5;

//for each robot, change these constants based on IR results
const int minIR[numIrSensors] = {605, 569, 640, 330, 565}; //robot #1
const int maxIR[numIrSensors] = {789,787,813, 629,792};
const int lineRange =70 ; // desired output value range (between 0 and 100)
const int nbSamples = 4; //window length for filtering
const int whiteThresh = lineRange/3;
const int blackThresh = lineRange/1.2;

class SensorsClass
{
protected:
    //int storage[numIrSensors][nbSamples]

public:
    //void calibrateIR();
    void readRaw(); //raw data (as seen by the ADC)
    void read(int tab[numIrSensors],bool filter, bool debug=0);
};
extern SoftwareSerial mySerial;

/* Sensor order :

----- front -----
-----0-----1-----2-----
-----
-----4-----3-----
-----flippers-----
*/

```

```
|| #endif
```


Listing A.5: *Sensors.cpp*

```

//
//
//

#include "Sensors.h"

// void SensorsClass::calibrate

//reads at 10kHz

//temp use for calibration
void SensorsClass::readRaw()
{
    mySerial.print(analogRead(IR1)); mySerial.print("\t");
    mySerial.print(analogRead(IR2)); mySerial.print("\t");
    mySerial.print(analogRead(IR3)); mySerial.print("\t");
    mySerial.print(analogRead(IR4)); mySerial.print("\t");
    mySerial.print(analogRead(IR5)); mySerial.print("\t");
    mySerial.println();
}

void SensorsClass::read(int ptr[], bool filter, bool debug)
{
    //always better to avoid float -> use signed long to avoid overflow
    //reinitialize tab each time
    int storage[numIrSensors]= {0};

    if (filter)
    {
        for(int j=0; j < nbSamples; j++)
        {
            storage[0] += ((signed long)analogRead(IR1)- minIR[0])*
                lineRange/(maxIR[0] -minIR[0]);
            storage[1] += ((signed long)analogRead(IR2)- minIR[1])*
                lineRange/(maxIR[1] -minIR[1]);
            storage[2] += ((signed long)analogRead(IR3)- minIR[2])*
                lineRange/(maxIR[2] -minIR[2]);
            storage[3] += ((signed long)analogRead(IR4)- minIR[3])*
                lineRange/(maxIR[3] -minIR[3]);
            storage[4] += ((signed long)analogRead(IR5)- minIR[4])*
                lineRange/(maxIR[4] -minIR[4]);
            delay(500);
        }

        for (int i=0; i<numIrSensors; i++)
            ptr[i] = storage[i]/nbSamples;
    }
}

```

```
else
{
    ptr[0] = ((signed long)analogRead(IR1)- minIR[0])*lineRange/(maxIR[0]
        -minIR[0]);
    ptr[1] = ((signed long)analogRead(IR2)- minIR[1])*lineRange/(maxIR[1]
        -minIR[1]);
    ptr[2] = ((signed long)analogRead(IR3)- minIR[2])*lineRange/(maxIR[2]
        -minIR[2]);
    ptr[3] = ((signed long)analogRead(IR4)- minIR[3])*lineRange/(maxIR[3]
        -minIR[3]);
    ptr[4] = ((signed long)analogRead(IR5)- minIR[4])*lineRange/(maxIR[4]
        -minIR[4]);
}

if (debug)
{
    mySerial.print(ptr[0]); mySerial.print("\t");
    mySerial.print(ptr[1]); mySerial.print("\t");
    mySerial.print(ptr[2]); mySerial.print("\t");
    mySerial.print(ptr[3]); mySerial.print("\t");
    mySerial.print(ptr[4]); mySerial.print("\t");

    mySerial.println();
}
}

//SensorsClass Sensors;
//SoftwareSerial mySerial;
```

Listing A.6: *Control.h*

```

// Control.h

#ifndef _CONTROL_h
#define _CONTROL_h

#if defined(ARDUINO) && ARDUINO >= 100
    #include "Arduino.h"
#else
    #include "WProgram.h"
#endif

//control variables constants and variables
const float axleLength = 145;
const float radius = 72;

const float k_rho = 700; // must be > 0
const float k_a= 6000; // must be > k_rho
const float k_b =-6000 ; // k_b < 0
//min abs distance (mm) before starting to climb

//scale the outputs within min and max
const int minSpeed = 200;
const int maxSpeed = 50;

class ControlClass
{
protected:
    float rho;
    float alpha;
    float beta;

    float v; // depends on rho
    float omg; // depends on alpha, beta

    float upBound;

public:
    void lyap(float , float, float, int tab[2], bool debug = 0); // laypunov
        controller
        // normalization done realtive to the maximum area from which a Tag is still
        visible
    void normalize(int tab[2], bool debug =0);
};

extern ControlClass Control;

#endif

```

Listing A.7: *Control.cpp*

```

#include "Control.h"

//computes a lyapunov controller from aprilTag information
//x, y in mm, roll in degrees
void ControlClass::lyap(float x, float y, float roll, int tab[], bool debug)
{
    rho = sqrt(x*x +y*y);
    //alpha is chosen such that it is positive when y negative
    alpha = -atan(y/x); // is the same as atan2 if x> 0
    beta = -alpha-roll/180*PI;// beta is replaced by beta-90 for new arrival
        position

    v = k_rho*rho;
    omg = k_a*alpha + k_b*beta;

    tab[0] = v/radius - omg*axleLength/(2*radius);
    tab[1] =v/radius + omg*axleLength/(2*radius);

    if(debug)
    {
        Serial.print(" alpha "); Serial.print(alpha*180/PI);
        Serial.print(" beta "); Serial.print(beta*180/PI);
        Serial.print(" v "); Serial.print(v);
        Serial.print(" omg "); Serial.println(omg);
        Serial.print(" sLeft "); Serial.println(tab[0]);
        Serial.print(" sRight "); Serial.println(tab[1]);
        //Serial.print(" upBound "); Serial.println(upBound);
    }
}

void ControlClass::normalize(int tab[2], bool debug)
{
    //corresponds to max perimeter
    int tempTab[2];
    lyap(410, 200, -10, tempTab);
    upBound = max(tempTab[0], tempTab[1]); //ouputs 8200

    if(debug)
    {
        Serial.print(" b4 normalizationLeft "); Serial.println(tab[0]);
        Serial.print(" b4 normalization Right "); Serial.println(tab[1]);
    }

    if (tab[0] >0)
        tab[0] = (1- tab[0]/upBound)*(minSpeed-maxSpeed) + maxSpeed;
}

```

```
else if (tab[0] < 0)
    tab[0] = -(1+ tab[0]/upBound)*(minSpeed-maxSpeed) - maxSpeed;

if (tab[1] > 0)
    tab[1] = (1-tab[1]/upBound)*(minSpeed -maxSpeed) + maxSpeed;
else if (tab[1] < 0)
    tab[1] = -(1+tab[1]/upBound)*(minSpeed -maxSpeed) - maxSpeed;

if(debug)
{
    Serial.print(" after normalizationLeft "); Serial.println(tab[0]);
    Serial.print(" after normalization Right "); Serial.println(tab[1]);
}
}
ControlClass Control;
```

List of Figures

1.1	(a)weaver ant tower [3]; (b) fire ant raft (pushed into the water with a twig) [1] ; (c) army ant bridge[2]; (d) weaver ant chain [3]; (e) Japanese honey bee oven around a hornet [2] .	5
1.2	modular robots (from left to right) : (a) lattice-like robot MICHE [5], (b) M Blocks [8] ,(c) two SMORES modules [10], (d) a complex assembly of SuperBot modules [11]	7
1.3	climbing robots for rough terrain (from left to right): (a) SHRIMP, (b) reconfigurable tank-tread robot, (c) LaMalice, (d) Mobit	7
1.4	ICM wind turbine inspection robot using vacuum to climb on brick, metal, or concrete [21]	8
1.5	electroadhesive pads implemented on a 1DoF robot	9
1.6	robot with passive suction cups	10
1.7	different ways of gripping and attaching : (a) bipedal robot with 2 grippers [35], (b) RiSe robot [39], (c) Treebot [34] , (e) GeckoBot [40]	10
2.1	overview of some of the climbing categories (rectangle) and some of the common-used mechanics for climbing (circles); the chosen path is highlighted in orange; literature references are added between brackets	12
2.2	first selection among mechanical designs; the marks are given from 0 to 5, where 5 solves the topic in the most satisfying way; the chosen design is highlighted in red	13
2.3	flipper-based design possibilities; the chosen solution is highlighted in red	15
3.1	simplified robot model; two conditions have to be respected in order to climb: the CoM has to reach the edge of the obstacle and the robot has to be in static equilibrium at all time	17
3.2	19
3.3	illustration of function (3.14) : necessary friction to climb with a given aspect ratio; all points in shaded region are valid	20
3.4	variation of the minimum coefficient of friction necessary to climb with respect to the aspect ratio $k = \frac{height}{length}$	20
3.5	optimal design line: for each offset of the CoM the last point of the minimum friction line is taken (left) and represented on a separate curve (right)	21
3.6	two possible structure configurations: either the robot can return to horizontal position as an intermediate step of the ascension, or it continues the ascension with the same or smaller orientation angle; the climbing robot is represented in red	21

3.7	simplification of climbing on several layers of robots; as in a standard inclined plane problem, the friction coefficient has to be larger than the tangent of the inclination angle for the robot to hold still	22
3.8	additional static condition on μ : figure 3.5a is represented with the additional condition (3.16) in dashed-dotted lines; the green dashed line represents the reunion between the two conditions; a robot can climb on another and on several layers of robots if its design is situated above the green dashed line	23
3.10	example of robot with flippers	23
3.9	performance comparison between robots with different flipper lengths; green line represents flippers of the same size as the robot, which makes the new total length twice the initial one, whereas blue line represents no flippers at all;	24
4.2	Tamyia track set from [43]	25
4.1	commented overview of the robot assembly	26
4.3	with a mass of 0.5 kg, a radius of 4.5 cm and a friction of 1.8	27
4.4	298:1 HP Micro Metal Gearmotor with extended shaft for encoders http://www.pololu.com/product/2208	28
4.5	forces exerted in an extreme situation: the full arrows represent the efforts on the robot body (excluding the flipper)	28
4.6	robot chassis with main characteristics	29
4.7	the worm drive support - illustrated in red	30
4.8	robot deck with longitudinal and transverse with the stripe for line following and edge detection	31
5.1	1st model built with a prudent aspect ratio of 0.37; only the minimum electronics for remote controlling are present; the flushed frontal design of the chassis does not allow it to tackle obstacles diagonally	32
5.2	illustration of climbing on an obstacle more than twice the height of the robot	33
5.3	second embodiment with an aspect ratio of $k = 0.7$	33
5.4	situation of the 2 designs with respect to the theoretical predictions; 1st design is represented with open and closed flippers, while the second only with open flippers (for visibility reasons)	34
5.5	ground clearance and side stuck position: most of the robot's weight is supported by red point (I) , which means there is not enough weight on (II) to ensure traction	35
5.6	different climbing configurations: with flippers backwards (I) and flippers forwards (Ia) ;	35
5.7	results in climbing from different angle in different configurations; 10 trials have been conducted for each angle (spanning from 0 to 170°); successes are indicated in blue, failures in red , and slides in orange; the <i>backwards</i> configuration is kept as definitive	36
5.8	phase (II) is meant to assess how often the robot would get stuck on the side (as sketched in 5.10 and phase (III) is meant to assess the performance of climbing on a 2-layered structure	37

5.9	results of phase II (climbing from a shifted position) and of phase III (climbing on 2 robot layers); in phase II, “difficult” means the robot gets temporarily stuck in a position similar to 5.10, and in phase III “misalignment” occurs when the robot falls before the second layer is reached	38
5.10	sliding on the edge of another robot can occur after climbing; although the robot is not stuck, the climbing has to be repeated, because it is not considered a stable position for further construction	38
6.1	some possibilities of self-assembled structures; the 2D tower is chosen	39
6.2	a Raspberry PI camera is used for navigation and alignment	40
6.3	bottom and top view of the robot; when on another robot, the robot mostly relies on its IR sensors; the longitudinal white pattern guides the robot and the transverse one indicates an imminent edge	41
6.4	overall schematic of the robot’s electrical components; black arrows are inputs and red are outputs; the communication protocol is indicated in italic	42
6.5	overview and placement of the sensors in the robot; the RPI board and batteries are omitted for visibility purposes	43
6.6	example of April Tag detection: for visualization purposes, a rectangle is drawn around the tag when detected; orientation information is then extracted using homography	43
6.7	position of the camera and of the April Tag on a robot	44
6.8	possible orders for a 2D tower with 6 robots: the first 3 steps are highlighted in red for better visibility	45
6.9	illustration of an arbitrary-sized structure formation algorithm for a 2D pyramid; the first 6 steps are shown; generally, the robot has to climb and count the number of robots beneath; from an algorithmic perspective, phase II is the same as phase IV, and III the same as VI; in phase V counting is not necessary as an April Tag is detected directly after climbing; this algorithm can be reiterated for an arbitrary-sized structure and stays the same for each robot	46
6.10	recording of phase II; the exact same strategy is iterated in phase IV	47
6.11	recording of phase III; the exact same strategy is reiterated on phase VI	47
6.12	recording of phase V	47
6.13	top-to-bottom code implementation scheme, from high-level behaviors to hardware interfacing	48
6.14	alignment with non-holonomic robots: ρ , α , and β have to be controlled to 0; x , y and γ are extracted using the camera, April Tags and homography	49
7.1	accepted error tolerance for completion of position III and VI ; dimensions of the robot are 183 x 180 x 73 mm	52
7.2	after the end of each assessment, an empty robot is placed at the same position the active one stopped previously	53

7.3 alignment errors (for position and angle) at real scale for phase IV ; a 25mm square is
drown for illustrative purposes; the red mark is the ideal position (when the robot is
aligned manually) 54

List of Tables

4.1	motor specifications	28
5.1	robot specifications	34
7.1	assessment of the 6 algorithm phases ; when available, the misalignment error relative to the ideal position is computed and the min, max and median values are listed	53