University of Aarhus
Department of Computer Science
Åbogade 34
DK-8200 Århus N, Denmark

Master Project

August 31st 2011

LUDEP

Leading Units & Drone Enabled Probing

Supervisor: Ole Caprani

Guillaume DEPOYANT ***20100192*** *guillaume@ludep.com*

Michaël LUDMANN ***20100194*** michael@ludep.com

ABSTRACT

Achieving spatial formation and position location in a flock of autonomous, mobile robots in a pervasive way, to explore a mostly unknown environment–where GPS may not be available our good enough - with a minimal setup and at a low cost, remains a challenge in the field of autonomous vehicles behavior, be it for personal, entertaining purpose or for scenarios like a rescue mission in hazardous conditions. This master project investigates the possibilities of creating such a system with inexpensive hardware available in stores, fully open-source code, in a short development time and that can be easily started up.

The project is mostly empirical and focuses on physical embodiment of the whole system, yet is partly supported with mathematical and physical models for specific tasks. The robots are built out of the last LEGO Mindstorms NXT kits, a rather simple and cheap robotic platform, and are designed as omnidirectional vehicles. Their localization is coordinated by an AR.Drone, the Unmanned Aerial Vehicle (UAV) sold by Parrot. Positions of ground units are reported thanks to custom image analysis detection performed on the video signal send by the vertical camera of the UAV, which tracks autonomously robots of the flock. A third, intermediate type of device, that is a computer, together with a robot of the flock, is in charge of transmitting data from the UAV to the flock while leading it and maintaining a specific formation. A formation is dynamic and evolves in real-time depending on the available units, following a set of rules. All mobile units use stabilization controllers.

This report presents the methods applied to fulfill the main goal, by splitting it into sub-problems which have a limited set of requirements before being considered as completed, and which always leave room for further improvements that are thoroughly discussed. Thus, the resulting system shows its efficiency and stability within a frame that has been gradually established with the project advancement, and provides a general framework whose parts can be updated separately. It also illustrates how two recent technologies, different in their principles, can be merged together and augment each other capacities in an innovative way.

PREFACE

Both coming from the same high school, coincidence led us to the same engineering school. It has been more than five years that we know each other now, three years that we are used to do projects together and that we share the same passion, namely robotics. But for the last year as students, we wanted to do something bigger. When we came back in Århus after Christmas holidays and had to choose a subject for our respective master projects, we asked ourselves: "How difficult could it be to create two projects and merge them together as a relevant and consistent one?"

Well, we really got excited by the idea. The AR.Drone seemed to be a really advanced device, pleasant to code or work (and play) with, so we took the leap. We thought about it over a few days in order to find an idea that could make sense, an idea that could interest the community and fascinate us at the same time and we came out with LUDEP: Leading Units & Drone Enabled Probing. With all the background we had and everything we learned when we attended Ole Caprani's course on Embedded Systems, we were ready to have some sleepless nights and try to make this ambitious project working.

We were really optimistic all along the coding of the project, we got to work with a lot of technologies, approached enriching problems but even when we encountered huge obstacles the projects met its best improvements. Our website and social channels made its own, humble mark in the community and we are really grateful to all the people that followed us, and encouraged us in what we did while they spread the word.

Guillaume & Michaël

TABLE OF CONTENTS

1 INTRODUCTION

Deploying flocks of robots in unknown environments, be they urban, hostile or hazardous, is often a necessary challenge for rescue teams that cannot have a reliable access to good GPS measurements, face communication issues, and may not be able to send human backup directly. Collaboration between aerial and ground units is a solution already in use in fields like defense, civil protection and emergency situations.

However, those systems are not always thought as a whole group cooperating in an autonomous way, and may lack a proper communication network that would enable a more consistent behavior with the least human help required. Such an interactive activity raises indeed many computer science questions, and physical and even mathematical issues as well – while it makes the team behavior more complex.

Besides, advanced technologies have become more widespread with time, to the point where the general public can now afford personal robotic devices with good hardware and programmable software to customize their behavior. This project will describe a framework for a collaborative system between air vehicles and ground robots which should be as autonomous as possible, while investigating an inexpensive solution that could be reproduced by anyone who would just have to buy the hardware in any regular store.

1.1 MOTIVATION

During the course *EMBEDDED SYSTEMS AND EMBODIED AGENTS*[1] we took at Aarhus University, we were introduced to a broad range of robotic topics, and among them were presented behaviors for individual robots with ideas of group collaboration and autonomous control. Since this was not a focus point in the semester projects we selected, we were interested in spending more time discovering ways of handling large groups of robots rather than independent units.

On top of that, one of us had recently bought a quadrotor aerial vehicle with the purpose of taking advantages of the hacking possibilities it offers to develop some applications involving other robots, like tracking a robot vehicle moving on the ground. Since this was something not achieved so far with this kind of hardware, we decided that this master project would be a good opportunity to study the feasibility of such a goal.

Ground vehicles like a flock of robots and aerial vehicles like a quadrotor being rather limited in their own ways when it comes to complete exploration mission, where probing a new area might be a goal, they could support each other well and team up for a better efficiency. Thus we agreed on willing to prove this concept and accept this technological challenge.

1.2 FOCUS OF THE PROJECT

This project will describe the setup of our whole system, where an unmanned aerial vehicle (UAV) is keeping track of a ground robot that leads a flock of similar robots. The UAV flies autonomously and uses its cameras to help the units moving in a defined formation, while communicating with the leader through an interfacing device. We will focus on establishing a stable framework that enables a proper

[1] *http://www.legolab.daimi.au.dk/*

communication between all involved elements, so they might still be upgraded separately, and even work in alternative setups (like with an usb webcam instead of the drone for instance).

This is a project so vast that it will always ask for improvements, thus we want to focus on a proof of concept upon which one can still build up and add features.

1.3 GOAL

The main goal is to create a working system and a proper network layer, which enables smooth interactions between an AR.Drone that is the quadrotor from Parrot, and robots built out of LEGO Mindstorms (using the NXT controller). A PC, or even a smartphone, will do the necessary interface between the drone and the flock; they have to be integrated in the whole system.

We want a dynamic flock that should adjust itself depending on the number of robots present on the ground, and take lost robots in consideration. A robot will be the leader of the formation; cases where it would have to pass its role to another robot have to be taken into account.

Another goal is to achieve a portable, open-source project, not too complicated to start, with non-expensive, easily available hardware, in order to be integrated in some robotics games or events.

1.4 METHOD

Three major steps will we taken in order to fulfill our main goals:

1. Investigating the possibilities of the drone and then proceeding to gain control over it, step by step. This will be achieved by looking at the current state of the art of drones control and applications, and more especially with the AR.Drone. Controllers will then be implemented to enable autonomous behavior and image analysis will be added as another layer.
2. Deciding on the best design for our robots that fits a flock behavior well, and justifying the choice. Then, implementing a reliable controller that allows smooth movements without much drifting, even while driving “blind”. Last but not least, establishing a framework for a dynamic flock, stabilizing connection with many robots and creating simple rules for a good behavior.
3. Putting the previous pieces together, to test the whole system. First, the image analysis part of the drone will be used with a static ceiling camera, in order to experiment with the flock alone. Then, the drone will be integrated with the flock.

The first two points are independent enough to be investigated separately, thus each of us will work on those two topics at the same time. The third point consists basically in merging the previous work and will be subject to lots of testing and tuning.

1.5 RESULTS

The project will illustrate how it met our expectations, insofar as multiple experiments proved the stability and the responsiveness of the whole system in our testing environment. In practical terms, we managed to get an AR.Drone hover on top of the selected robot leader, and change accordingly depending on which robot is the current leader, while our flock of four identical omni-directional robots moves dynamically depending on the position and direction of the leader and keeps its formation, in real-time and without noticeable delay. Yet, some topics leave possibilities for further improvements, which were not always doable in the context of our precise setup because of physical

constraints (test room sometimes too small for the drone, strong electromagnetic fields disrupting our compasses) – even if we eventually achieved to deal with it by simplifying the problem.

1.6 OVERVIEW OF THE PROJECT

Part one of this project will contain all the work achieved on the drone, from scratch to the real-time, in-flight image analysis by color detection, and will dwell in length on the implementation of a good flight controller.

Part two will cover the building and designing of the omni-directional robots that will be part of the flock. It will focus on the best ways to control them, with mathematical models and physical algorithmic controllers. Implementation of a general framework for the flock will close this part.

Once the basis of our work has been solidly established, part three will describe how the real purpose of our project is achieved by merging the drone and the flock behavior together. Further results and analysis of the work will be provided.

1.7 WEBSITE

This report is not fully complete without the website we built alongside this project: *ludep.com*.

The most technical parts are explained in every one of our articles that are available online, and they are to be considered themselves as small reports describing specific sub-problems of the project. Furthermore, a website sounds like the best way to clearly explain what is at stake in such an empirical project, since we can take a lot of video and pictures illustrating our tests and results, all posted on our pages. It eventually became a good way to get feedback from a community enthusiastic about the progress of our work, on top of providing a documentation we hope to be solid for other developers or curious people interested in the implementation.

Those are the reasons why this report merely summarizes our work and should be read with the corresponding webpages together. All our articles are mentioned in the report when we consider that the reader might want to know more about a specific part, and the matching links are provided consistently.

2 DRONE

Note to the reader: This whole chapter, and the next two as well, are presented in a way as to encourage the reader to go back and forth between the report and relevant articles we wrote alongside the advancement of the project. That is the reason why topics approached in this report might sometimes appear vague and do not go into full implementation details. One could see it as an easy-to-understand introduction and overview to more complete articles, which hold the needed references inside, and are all available on the project's website: *ludep.com*.

Every time an article is summarized in this report, its name and the full link to its location are provided. Those articles are a fair representation of the chronology of the project, and present plenty of videos, pictures, illustrations and references that are as important as the remaining of the report.

2.1 MOTIVATION

Sending a flock of robots (discussed in the next chapter) anywhere and being able to track it and help it keep its formation are the key points of the project. Tracking robots outdoors can be achieved by means of Global Positioning System (GPS) chips carried by each unit. This solution, as we have learned during our Pervasive Positioning course, presents however major drawbacks:

- It works almost exclusively outdoors, very rarely indoors (depending on the building structure and its surroundings), and has accuracy issues, especially while being surrounded by high building or trees. On a small scale, with a few robots, an accuracy oscillating between 1 to 3 meters in best cases scenarios is clearly not enough to distinguish two robots close to each other –that is if we use the GPS signal available to regular users who owns a GPS device.
- It requires a rather long time for the first "cold fix", that is when the GPS is turned on for the first time, it may be necessary to wait for a few minutes before getting a steady signal.
- If no fix can be established, or if units are passing through an area without any GPS signal, then they are lost. Dead-reckoning solutions, currently experimented on smartphones, exist to reduce this effect temporarily, but still, a loss in performance is to be expected.

Indoor Positioning System (IPS) also starts to get better and more affordable, but they require a pre-defined setup in an environment previously known by the users and thus the units: mapping the place is needed, as well as setting many sensors in the indoors surroundings (ultrasound emitters and receivers, RFID tags…). Calibrating it is time consuming, and it discards all possibilities of flexibility, and deploying the system anywhere else is done with a lot of added cost.

As a contrary, taking advantage of the embedded hardware and software capabilities of a drone (a flying, steady, stable quadrotor with high maneuverability thanks to its small size, that enables indoors flight) seems relevant for solving most of the above-mentioned problems. As soon as you get a vertical camera pointing toward the ground, then you can simulate a local coordinates system, where units can be positioned relatively to each other by image analysis. An absolute positioning system can even be added, if this is really required, with only one GPS chip on the drone, to locate all the flock – circumventing also the issue of accuracy and time of cold fixes, since the altitude of the drone would greatly improve signal readings. Image analysis of the video signal could also be a solution to reckon the surroundings.

Finally, the drone we plan on using is a really inexpensive device[2], available in many stores to anybody who want to buy it, quite recent, and with a growing community of passionate developers who explore together the vast potential of the device. Moreover, it is a good excuse to get our hands on a high-tech product, with which everything is yet to be done. This also means that some documentation will be missing, and, overall, it creates a good technological challenge since we cannot allow ourselves to rest on lots of some previously acquired laurels…

Note that the AR.Drone may be referenced later under other more generic names: drone, UAV (Unmanned Aerial Vehicle), quadrotor, quadricopter.

2.2 PRESENTATION OF THE AR.DRONE

2.2.1 HARDWARE

FIGURE 1: TOP VIEW OF PARROT AR.DRONE, WITH ITS INDOORS SHIELD

The drone sold by Parrot, as it stands on Figure 1, is an inexpensive UAV with great embedded technology that came out late August 2010. Its main features are:

- One VGA front camera (640*480 pixels, 93° angle)
- One CMOS vertical camera (176*144 pixels, 64° angle)
- Ultrasound altimeter for vertical stabilization
- Communication over WiFi connection for navigation data and video transmission
- Inertial guidance (accelerometer, gyrometer) with security system
- Autopilot maneuvers (take-off, landing, hovering)
- Embedded 2D tags and 3D beacons detection for augmented reality
- 12-15 minutes flying time provided by one Lithium Polymer battery (1000mAh, 11.1V)

2.2.2 SOFTWARE

This drone comes with an Application Programming Interface (API) in C/C++ that eases the programing of devices interfaced with it. So far, Parrot did not provide developers with the

[2] AR.Drone by Parrot, around 300€ (2500DKK) in shops or online. Came out in August 2010.

necessary tools to access and change the drone's firmware to make a custom one. However, it is possible to do everything to control the drone remotely and get feedbacks from all the sensors and components.

Creation of new applications is possible on many operating systems: Windows, Linux, Android, Mac and iDevices (iPhone…). The one major requirement is to be able to connect with the programmed device on the drone's hotspot WiFi in ad-hoc mode (which is rarely enabled by default on an Android device, but otherwise mostly possible)[3].

A MORE DETAILED OVERVIEW OF THE DRONE PRESENTING ITS PHYSICS, HARDWARE PRECISIONS, SOFTWARE SETUP, AND HOW TO ACCESS NAVIGATION DATA IS AVAILABLE IN THE FIRST OF OUR APPENDICES.

2.2.3 STATE OF THE ART

Starting a new project from scratch presents the risk to redo something that has already been made by another team of developer/scientists/enthusiasts. And even if the hardware may not quite exactly be the same as the one we plan to use, it is worth having a look around the fields of robot tracking, image analysis with a quadrotor, and advanced autonomous behaviour of a flying device.

Some time was thus spent on looking for the state of the art on those topics. We found some great ideas with rather impressive results of actions performed by the AR.Drone itself or by more custom Do-It-Yourself UAVs. Our article *GETTING SOME INSPIRATION*[4] deals with interesting examples and informative links.

However those ideas are never about a drone whose purpose is to follow a moving pattern on the ground in a steady and continuous fashion, not to mention the assistance of a whole group of robots. Actually, one of the two (publicly available – we can imagine that the military investigated at least once this topic) references that deals with one aspect of the project was performed by people working in a NASA research centre facility. They precisely used the very same AR.Drone to follow a remote controlled vehicle on the ground. Yet there is a major difference in the way their solution has been designed and implemented: the system operates indeed with multiple surrounding, stationary cameras that do motion tracking of both the drone and the vehicle. While this is efficient in this setup, it lacks the mobility we would like to achieve. Some work may still then be done in this direction.

Further details about this NASA project and our analysis can be read in our article *SAME IDEA, DIFFERENT APPROACH*[5].

The other reference was found later[6], during the course of our project. People from ELIOS Lab in Italy are currently working on organising a robot team to undertake a search and rescue mission in urban environments. They use the same hardware as we do: LEGO NXT Robots and an AR.Drone. However, their goal is different and the solution implemented in another way: they send an UAV out to search for specific objects and get information on the field – and later, ground units are sent independently to go alone to the selected point, while using data processed by the probing drone. On the contrary, we

[3] This changed recently, toward the end of our project: the newest firmware (1.7.4) makes the AR.Drone start its own access point, meaning that any wireless enabled device should now be able to connect to it.

[4] *http://www.ludep.com/getting-some-inspiration/*

[5] *http://www.ludep.com/same-idea-different-approach/*

[6] *Cai Luo and Alessandro De Gloria, Designing an Air to Ground Robot Team using Agent-based Technology*, in European Research Consortium for Informatics and Mathematics (ERCIM) News. ELIOS Lab University of Genoa, Italy.

send the whole flock together with the UAV, and they explore the unknown altogether. Nonetheless, it is really stimulating to observe that our two teams came out separately with the same general kind of idea and hardware for prototyping it, while still going in two different directions.

2.3 GETTING CONTROL OF THE DRONE

Before getting our hands on some heavy coding, we first needed to learn how controlling the drone is made possible by other applications, and how we could consider starting to develop our own.

2.3.1 THIRD-PARTY APPLICATIONS

Already existing applications are necessary for the user who simply wants to experiments with the controls of the quadricopter. It is however no easy task when it comes to try that outside the beaten paths developed by Parrot (basically, thanks to one of their two applications available only on the Apple Store so far).

We therefore took some time to test third party applications on different platforms, in order to select the operating system which seems to most pleasant and easy to use with the drone. We were successful with one promising application on an Android smartphone, and a rather good program running on Windows.

The solutions we found to make everything work, as well as screenshots, are described in our article *TESTING THE DRONE WITH THIRD-PARTY APPLICATIONS*[7].

2.3.2 FIRST AD-HOC TRACKING TEST

Once we had everything in our hands to start some serious thinking about designing and implementing our system, we investigated the potential of the drone and its embedded software. Without much work, and with some thinking outside the box trick, we quickly managed to run an experiment where one moving robot on the ground is slowly followed by the flying drone.

For this to happen, we took advantage of the stabilizing feature auto-enabled on the drone while hovering: the vertical camera tracks moving points on the ground and helps the drone adjust its position accordingly (in case of external disturbance, like a gust of wind), in order to try hovering on top of the same spot. If the drone is stable, the ground made of a uniform color and a robot is moving, then the drone is “tricked” into “believing” that the robot is its reference pattern. A video illustrates this concept in our article *FIRST EXPERIMENTS*[8].

2.3.3 BASIC CUSTOM FLIGHT CONTROL APPLICATION

The AR.Drone is still a work-in-progress development platform, which means that its API and firmware get frequently updated – and developers have to adapt if they want to enjoy new functionalities or bug correction. When our project started, it was not really obvious which firmware has to be selected, or which operating system was best to develop our own application. For instance, the most recent firmware brought new possibilities while still being unstable, and using it with the Windows Software Development Kit (SDK) provided by Parrot implied losing the video stream.

[7] *http://www.ludep.com/testing-the-drone-with-third-party-applications/*
[8] *http://www.ludep.com/first-experiments/*

Linux quickly appeared to be more optimized for development, for it was less CPU-consuming and provided a good video support. We quickly built a Graphical User Interface (GUI) using an example provided by the SDK, and did not have to do much to get a nice navigation interface with plenty of data refreshing in real time.

It was also already possible to remotely activate the embedded roundel detection algorithm, which keeps track of roundels on the ground (see Figure 2 for a picture of a roundel). Coordinates were reported on the screen in real-time. This detection algorithm was not developed by ourselves, but rather by the developers at Parrot, the only people who have access to the firmware of the drone.

FIGURE 2: A ROUNDEL IS A TAG THAT MAY BE DETECTED BY THE EMBEDDED SOFTWARE ON BOTH CAMERA SIGNALS

Please note that some of those comments were true at the time we started our project, and that firmware, SDK and API have evolved in the meantime, probably correcting parts of problems we experienced. As for the technical details and other examples, they are provided in the article *CONTROLLING THE DRONE WITH OUR OWN BUILT PROGRAM - ROUNDEL RECOGNITION*[9].

2.3.4 HARDWARE ISSUES

Dealing with physical devices always involves some kind of hardware failure from time to time, especially with a flying object that is frequently subject to fall or crash into obstacles. Fortunately, the AR.Drone is conceived to be easily repairable, and spare replacement parts can be bought individually when one is broken. Fixing the drone is then a matter of minutes, which is really good to know if it were to be part of some event, or big shows with day-long presentations. We never needed to buy a complete new drone, even if it sustained plenty of crashes.

Besides, it has a really sturdy structure, and indoors shields are really efficient. Overall, this is a pretty stable device that mostly fails because of human error or bad software programming. Internal close-ups of the drone during reparation are available in our article *THE DAY WE CRASHED THE DRONE*[10].

2.4 TRACKING A TAG ON THE GROUND

After having been introduced to the possibilities of our drone and the different ways of coding it, we settled on a platform (Linux – Ubuntu), a firmware (v1.5.1) and a SDK version (v1.6, and v1.7 a few months later). It was then time to start designing a basic C++ program upon which we could build more and more later on.

Putting everything together in the same program, in a clean and proper way, is not an obvious task: one has to initiate the communication with the drone, handle reception and display of video signals, make sure to get all necessary navigation data in the right place, accept manual user input by means of

[9] *http://www.ludep.com/controlling-the-drone-with-our-own-built-program-roundels-recognition/*
[10] *http://www.ludep.com/the-day-we-crashed-the-drone/*

a game controller, and eventually implement a loop, preferably isolated in the code, where all our personal algorithm may be put to develop the autonomous behavior needed.

This is a task easier said than done, and it was absolutely necessary to achieve it correctly. Once this was the case, we were indeed able to focus much more on our detection and tracking goal, without having to bother with others aspects of the drone's control.

2.4.1 ROUNDEL TRACKING, PID CONTROL AND IMAGE ANALYSIS

2.4.1.1 DRONE INPUTS

The drone is controlled thanks to high-level commands that will rotate the rotors accordingly, and this is mainly done by changing four parameters:

- pitch: to go forward (negative values) or backward (positive values)
- roll: to go right (positive values) or left (negative values)
- gaz: to go up (negative values) and down (positive values)
- yaw: to turn right (positive values) and left (negative values)

Those values range from -25000 to +25000: the greatest their absolute value, the fastest the movement. To track an object on the ground, the drone can stay at the same altitude (that is performed by a simple proportional and derivative controller) and does not need to turn on itself, since it can move in any direction on the same 2D plane only with pitching and rolling (yawing is mainly a good way to point the horizontal camera in a specific direction). We therefore need to concentrate solely on the pitch and roll parameters.

2.4.1.2 ROUNDEL TRACKING

Using the embedded algorithm investigated in chapter *Basic custom flight control application*, it is possible to get the coordinates of a roundel on the ground (cf. Figure 2) and build a simple proportional controller (P-control) to start tracking it. The furthest the drone is from its goal, which is, having the roundel located in a square centered in its vertical camera video field, the fastest it will activate its rotors to correct the error. Our first rough results with this approach can be seen in videos in our article *CODING WITH THE DRONE – PERFORMING ROUNDEL TRACKING*[11]. It raised the necessity of implementing also a controller for the altitude.

2.4.1.3 PID CONTROL

The previously mentioned P-control that was implemented was not satisfying enough, thus the necessity to go one or two steps further, with a more elaborate controller that is a Proportional Integrative Derivative (PID) controller. Such a controller requires defining a way to measure the main variable parameter that is the error. In this system, the error is the distance from the object to track to the center of the field of view (FOV) of the vertical camera, insofar as the goal is to have the camera (i.e. the drone) precisely on top of the tracked object. We first worked in a Cartesian system of coordinates, before eventually moving on a more intuitive way to measure the error thanks to a polar coordinate system. In this representation of the FOV, the error is simply the radius length that characterizes the position of the roundel, as illustrated on Figure 3.

[11] *http://www.ludep.com/coding-with-the-drone-performing-roundel-tracking/*

$$pitch = -\sin\left(\theta * \frac{\pi}{180}\right) * speed$$

$$roll = \cos\left(\theta * \frac{\pi}{180}\right) * speed$$

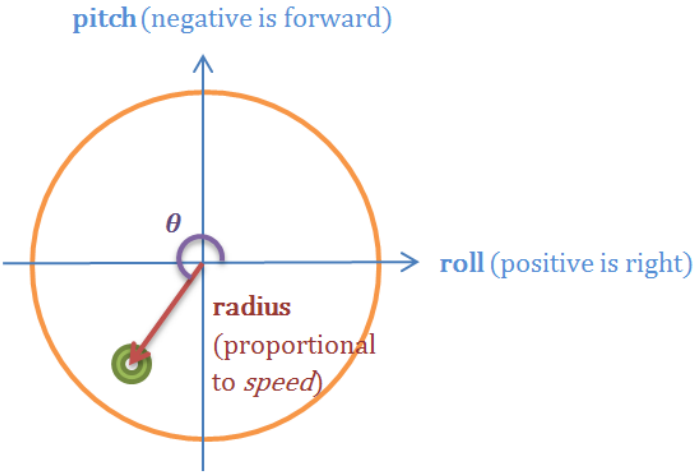


FIGURE 3: PID ERROR MEASUREMENT IN A POLAR SYSTEM OF COORDINATES, AND CONTROL VALUES ACTUALLY SENT TO THE DRONE.

Implementing the PID control in our system was then done like any regular PID controller, with three values (the proportional, integrative and derivative values) that are computed every time the drone gets a new frame on the video signal, and their sum provide a value directly sent to the motors, which are then fully controlled by this loop. This chapter does not dwell on the physical and mathematical part of a PID controller – but the concept is well explained in our articles which also provide additional references. The computation algorithm that loops as fast as possible is really straightforward, as showed below:

```
radiusError = radius; //proportional part; we want radius = 0
radiusIntegral = dampen * radiusIntegral + radiusError; //integral part
radiusDerivative = radiusError - radiusLastError;       //derivative part
radiusMove = self->kp * radiusError +
                     self->ki * radiusIntegral +
                     self->kd * radiusDerivative;
radiusLastError = radiusError;
```

`Kp`, `ki` and `kd` being the gains of the controller. They are set-up once and for all for a given system, however their tuning is the most difficult part in establishing a good working PID. This was achieved with systematic logging and graph analysis of the motors responses depending on the input gains.

Two of our articles discuss in more details the aspects to consider in a good PID:

- The second part of *PERFORMING SIMPLE IMAGE ANALYSIS AND FULL PID CONTROLLER WITH THE DRONE* [12] deals with a first PID in a Cartesian system, how it was tuned, and how the results might be plotted.
- *NEW PID WITH POLAR COORDINATES AND HOW TO IMPROVE REACTIVITY AND ACCURACY*[13] explains mathematical transformations and implementation of this second PID.

[12] *http://www.ludep.com/performing-simple-image-analysis-and-full-pid-controller-with-the-drone/*
[13] *http://www.ludep.com/drone-new-pid-with-polar-coordinates-and-howto-improve-reactivity-and-accuracy/*

2.4.2 IMPROVING RESPONSE-TIME

At some point, it was deemed necessary to implement our own detection algorithm (see below, *Image analysis*), and not to rely anymore on the embedded detection for the roundels. This change induced new issues due to the image analysis that is therefore done remotely in the case of a custom detection.

Thorough experiments were developed to draw the conclusion that it is not possible to do better than the embedded algorithm because of speed of transmission issues and loss of random frames as well. In fact, we established that a custom algorithm records 1.45 times less workable data than the embedded algorithm, for the very same tracking tests. Explanation of the experiments, schematics about the way data are processed, results and performances are available in *NEW PID WITH POLAR COORDINATES AND HOW TO IMPROVE REACTIVITY AND ACCURACY.*

The problem is that we really need our own custom detection algorithm to detect whatever object of the flock we want, and to distinguish them – which boils down to having to deal with this lag and exploring other possible improvements that would decrease the effect of the issue, even if it means having a drone that is not perfectly hovering.

2.4.3 INCREASING ACCURACY IN MEASUREMENT: CONSIDERING THE PHYSICAL WORLD

Part of our previously mentioned problem is solved, or at least the system is considerably improved, when the tilting of the drone is taken into account. If the drone is located on top of the same spot where there is a roundel, the coordinates returned will indeed vary more or less depending on the tilt angle. The greater the tilting, the bigger the offset. And a PID controller cannot behaves well if its core principle, that is the parameter measured which has to be corrected, is changing in an unexpected way because of the results of the PID correction. Figure 4 shows how tilting the vertical camera distorts the coordinates system on the ground. Besides, the roundel is clearly not at the same location as is it in a perfectly vertical situation when it is viewed from the drone viewpoint, whereas the drone and the roundel have not moved in the horizontal plane.

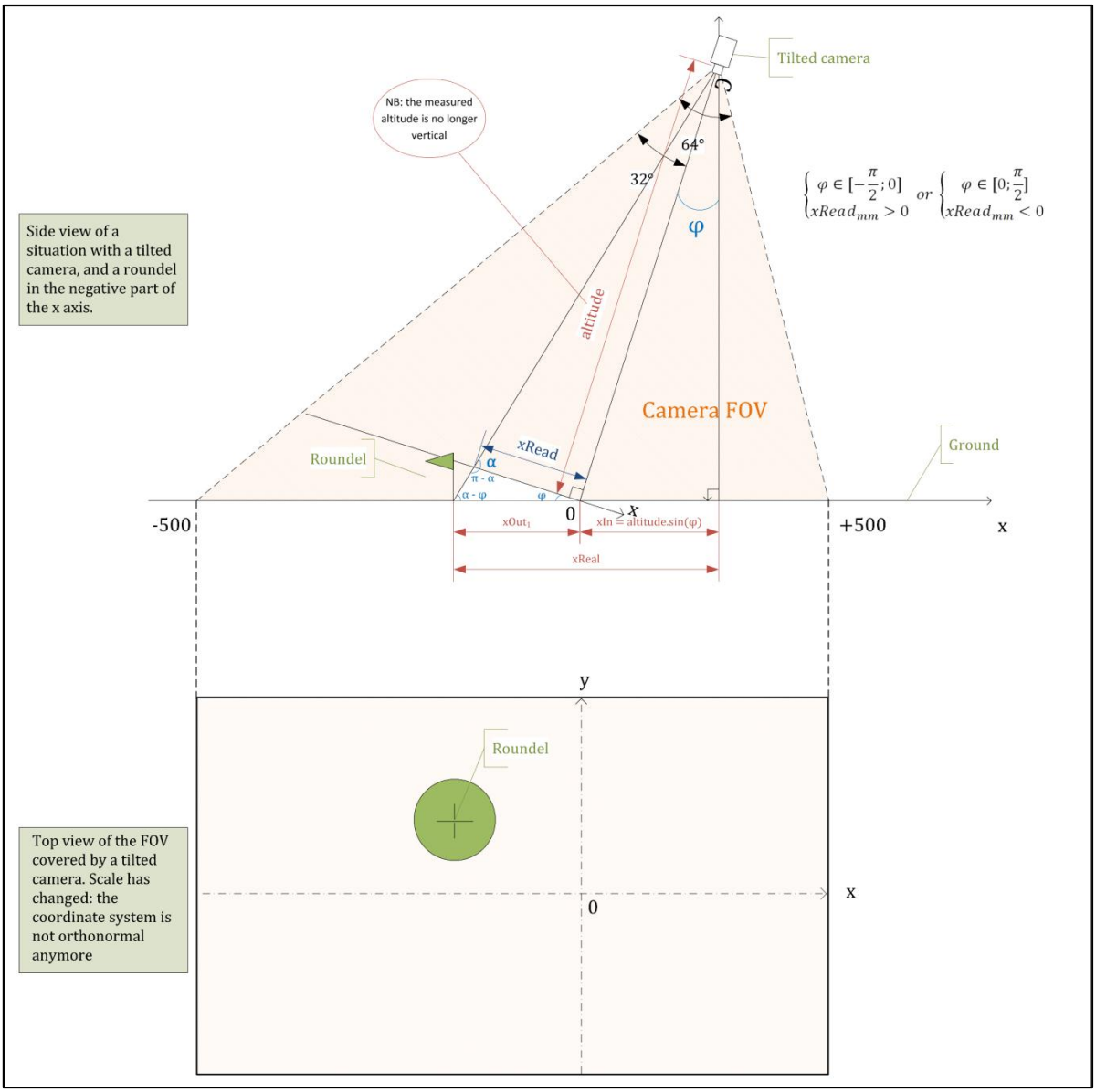


FIGURE 4: TILTED CAMERA WITH THE ROUNDEL TO THE LEFT OF THE FIELD OF VIEW (X<0).

This issue is worth considering and calls for some geometry calculus that lead to a positive leap in terms of performances, as illustrated in Figure 5. The article *TRACKING ALGORITHM, CONSIDERING THE INCLINATION OF THE DRONE*[14] digs extensively into this problem, which raises good questions about dealing with a physical world and sometimes hardware (sensors) limitation, and even proposes an approximated model that was tested and proved to be efficient.

[14] *http://www.ludep.com/tracking-algorithm-considering-the-inclination-of-the-drone/*

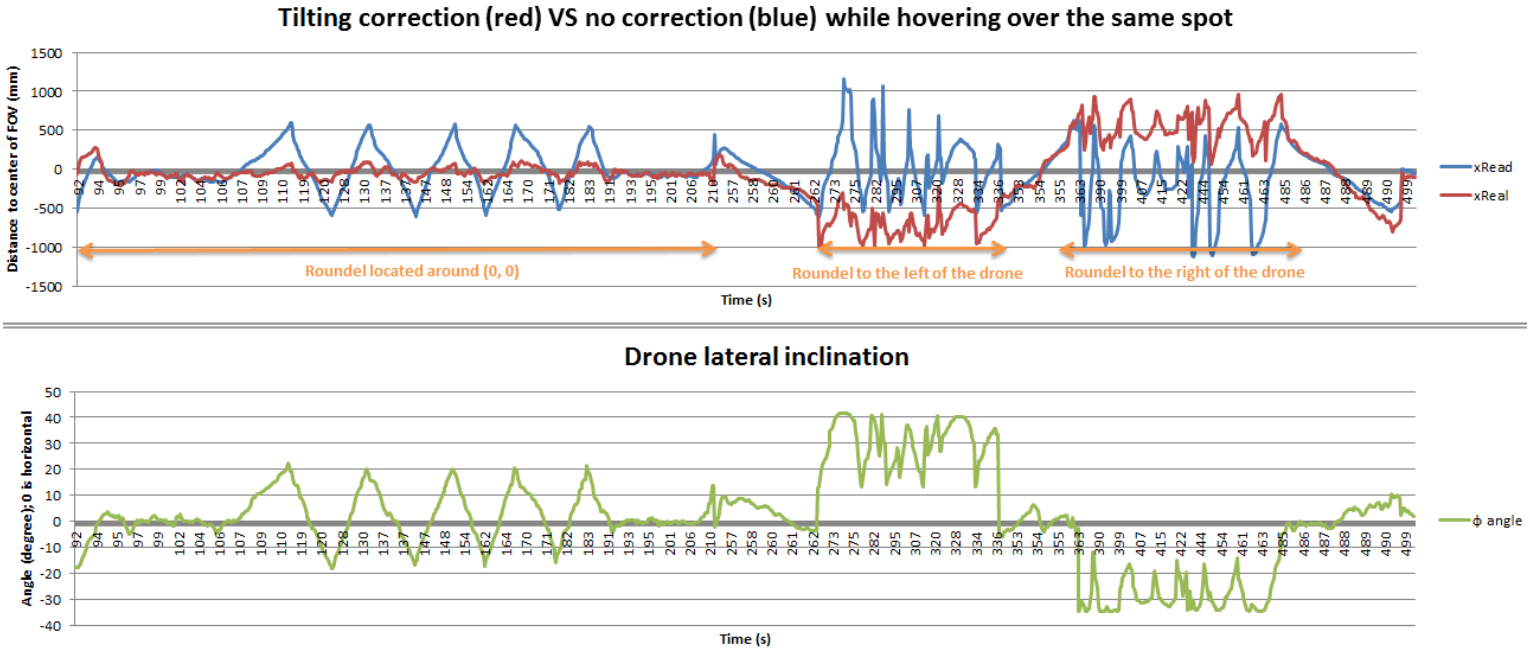


FIGURE 5: EFFICIENCY COMPARISON BETWEEN A CORRECTION OF THE TILTING AND A LACK OF CORRECTION: THE ROUNDEL WAS NOT MOVING, AND THE DRONE WAS SIMPLY STAYING OVER THE SAME POSITION, WHILE BEING GRADUALLY TILTED. IF THE DRONE IS TILTED (SEE IN GREEN) AND THE CORRECTION GOOD, THEN THE ROUNDEL IS SUPPOSED TO STILL BE DETECTED AT THE SAME LOCATION (SEE IN RED ON THE GRAPH).

2.5 IMAGE ANALYSIS

With the objective of tracking a whole flock of different robots with the drone, the solution of using the embedded roundel detection could not be more than temporary. Once done with the drone stabilization, it was then possible to focus on the image analysis problem itself.

2.5.1 CUSTOM SINGLE ROUNDEL RECOGNITION

To see if it was first possible to achieve on a remote device (e.g. a computer) what the AR.Drone can do by itself, custom roundel detection was investigated. The OpenCV library seemed to be the logical way of doing our own video analysis: it is open source, greatly supported by developers and big companies such as Intel, the library has more than 500 optimized algorithms that do not need to be reinvented, and it has been primarily developed for a C++ use, the language with which we develop our application for the drone.

A Canny detection and a Hough transform are enough to efficiently detect any circle on an image, which is perfectly adapted for roundels. It was measured that this custom algorithm performed even slightly better than the embedded algorithm in our setup, on the same frames (If the missing frames - that the algorithm do not receive due to hardware issues- are not taken into account). Unfortunately, as already mentioned in *Improving response-time*, the great number of missing frames decreases the added benefit in such way that it appears as less efficient. *PERFORMING SIMPLE IMAGE ANALYSIS AND FULL PID CONTROLLER WITH THE DRONE*[15] provides an excerpt of the code used for the functions that analyzes the frames, and a video illustrating its performances in real-time as well.

[15] *http://www.ludep.com/performing-simple-image-analysis-and-full-pid-controller-with-the-drone/*

2.5.2 MULTI-TAG DETECTION

Once the simple object detection was proved to be possible in our specific setup, with a moving camera and a WiFi connection, a more elaborate solution was sought to track multiple objects (i.e. robots of the flock) that have to be differentiated (e.g. robot #1, robot #2, and so on…).

Color detection and tracking, which look for specific colors in an image, was chosen. This method removes all the colors that are not wanted, so as to only keep the color we need to locate, by applying different color filters on each image. This is frame-independent, i.e. it does not require keeping a trace in memory of what happened before, which is a positive aspect since frames may have almost nothing in common depending on the movements of the drone. It is however a surrounding-light-sensitive solution, which means that results will vary depending on the daylight and lights that are turned on. So tuning might be often required before a new experiment.

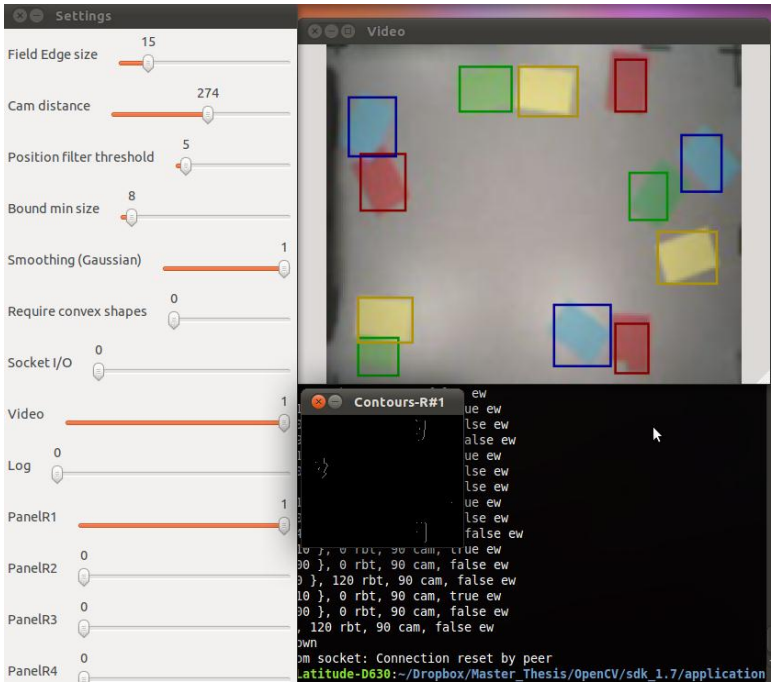


FIGURE 6: DETECTION OF MULTIPLE OBJECTS AT THE SAME TIME. THIS IS OUR MAIN INTERFACE THAT ENABLES US TO TEST AND SEE WHAT HAPPENS IN REAL-TIME, BY PROVIDING TOOLS TO TUNE PARAMETERS "ON THE GO". THE LEFT WINDOW IS THE USER INPUT INTERFACE, THE TOP RIGHT WINDOW DISPLAYS THE VIDEO WITH A MATCHING COLORS OVERLAY (COLORED RECTANGLES), AND THE SMALL BOTTOM RIGHT WINDOW CURRENTLY DISPLAYS THE EXTRACTED CONTOURS CORRESPONDING TO OUR RED COLOR.

Figure 6 presents the result that was achieved for this multi-objects detection purpose with an algorithm linear in time and constant in space consumption. It first identifies the colors that are expected to be in the field of view, by applying a color mask for each color, and then extracts the contours of the identified color-shapes. Even if it calls for other improvements, like being less light dependent, it is possible to keep track, without noticeable delay, of different colors at the same time.

The whole process is explained from scratch in our article *IMAGE ANALYSIS: COLOR DETECTION FOR MULTIPLE ROBOTS*[16], illustrated and explained through code snippets, screenshots, video, experiments and results.

[16] *http://www.ludep.com/image-analysis-color-detection-for-multiple-robots/*

2.6 CONCLUSION

The AR.Drone is a very satisfying and pleasing piece of hardware to work with, considering its nature. Despite a heavy use, mostly indoors, and crashes that are not to be counted anymore, this is a pretty reliable platform, whose only biggest flaw lies in its battery life (12-15 minutes). This is a serious issue that may be lessened by buying special batteries not supported by Parrot, which would multiply by two or three the battery life. Working with five official batteries and four chargers was enough for our developing time, but may be insufficient for a show presentation. In this case, we could quite easily think about using many drones, which would take relays, depending on their battery state: one flies over the flock, and once its battery is ¾ empty, another one takes off from the base and reaches the first one (embedded horizontal tag detection already enables the localization in space of another drone), which would then go back to the base to refill its battery.

Other than that, the drone's part of the project has been accomplished in its great lines: it is able to track and hover on top of a colored or circle- shape on the ground, while reporting the coordinates of many other tags in real-time. Managing such a reactive device that demands fast and efficient controllers which leave little room to imprecisions in this real-time setup was one of the biggest challenges to address. Exploring the field of video-analysis was the second hard part to investigate, which ended to provide results better than expected in terms of smoothness.

3 FLOCK

3.1 MOTIVATION

The challenge in that part of the project was to be able to create a flock knowing that the only information we could use would be a set of coordinates for different patterns on the floor. At this point the project could be split in two different parts:

- **Robots part:** the way we designed our robots, what are their inner mechanics, how to control them. We will justify why we chose the omnidirectional design instead of a normal one. The objective was to have a robot that could respond accordingly to our command lines. That way, we could easily switch the robots *(if we wanted to)* and/or the embedded program as long as they can understand the commands.
- **Flock behavior handling part:** how we handled the whole flock, how we defined it, what behaviors we implemented in order to bring this concept close to our expectations. We will expose why we chose our own method rather than others already existing.

3.2 ROBOTS

3.2.1 HARDWARE

As we followed the embedded systems course, we wanted to use the Lego Mindstorms NXT brick (Figure 7) because it is open source hardware and we were used to work on it. The three ports for the motors seemed to be exactly matching our needs *(c.f. next section)* and the four sensor ports available would allow us to give our project many openings.

FIGURE 7: LEGO MINDSTORMS NXT BRICK

3.2.2 SOFTWARE

There are many different firmware available and different languages to choose in order to code on the NXT brick (there is a quite complete on the article about the NXT on Wikipedia[17]). Once more, we chose the one we worked with the previous semester: *LeJOS NXJ.*

It is a high level open source language similar to Java, using a custom firmware developed by the LeJOS team. On top of having a complete and useful set of tools making communicating, uploading and coding easier, LeJOS offers the following:

- *Object oriented language (Java)*
- *Preemptive threads (tasks)*
- *Arrays, including multi-dimensional*
- *Recursion*
- *Synchronization*
- *Exceptions*
- *Java types including float, long, and String*
- *Most of the java.lang, java.util and java.io classes*
- *A Well-documented Robotics API*

3.2.3 WHY OMNIDIRECTIONAL?

Using omnidirectional robots was one of our first decisions. The previous semester, we had to choose a project to achieve for the embedded systems course and had some serious interests about using omniwheels. We did not do it after all; but for this project, we knew that we were about to ask robots to change their directions quite often according to the leader's moves: the omniwheels seemed to be a must for our purposes.

Omniwheels are wheels with small discs around the circumference which are perpendicular to the rolling direction[18]. Using several of them in a relevant fashion on a vehicle can enable it to be omnidirectional: it means that the vehicle can go in every direction with no need to turn. And this could be a real advantage because every robot would not spend any time in changing direction but would move directly instead. A lot of projects consist in creating omnidirectional structures or create a flock behavior, but according to our researches, coupling the two ideas is really unusual and we would be among the first one covering such a project.

3.2.4 OUR OWN ROBOTS DESIGNS

The NXT brick offering three ports for connecting motors, we decided to adopt the kiwi drive: it consists in using three omniwheels with each of their rotation axes in the same plane, making an angle of 120° with each other (triangular form) as you can see on Figure 8.

[17] Wikipedia article for LEGO Mindstorms NXT: *http://en.wikipedia.org/wiki/Lego_Mindstorms_NXT#Programming*
[18] Wikipedia article for omniwheels: *http://en.wikipedia.org/wiki/Omni_wheel*

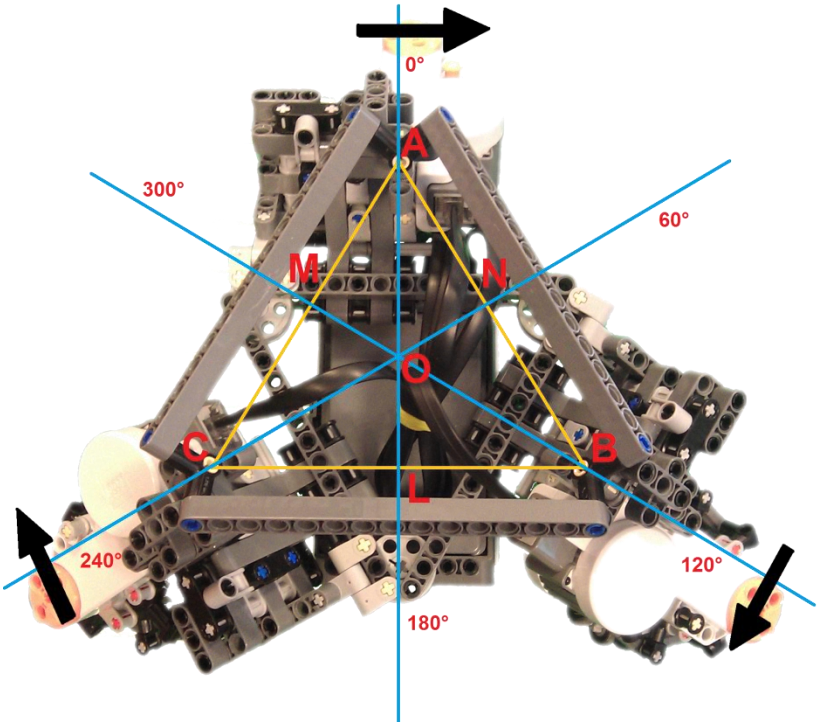


FIGURE 8: OMNIDIRECTIONAL ROBOT DESIGNED FOR KIWI DRIVE

We changed our design several times all along the projects. A reliable design must be as solid and light as possible without neglecting the fact that the center of gravity being low is a stability factor and the structure has to enable a perfect rotation of the omniwheels.

For the history of the designs, how and why we upgraded it, those articles explain it all:

- *GROUND UNIT CONSTRUCTION AND STRUCTURE IDEAS: IMPLEMENTING OMNIWHEELS*[19]: pictures and videos of our first design; tests with Android applications for controlling motors in order to move the robots in a rough way;
- *IMPROVING THE OMNIDIRECTIONAL ROBOT DESIGN*[20]: why the previous design needed to be improved and the omniwheel design as well;
- *ARRIVAL OF THE NEW OMNIWHEELS*[21]: why we had to switch to the Rotacaster[22] omniwheels, the difference between them and our home made design in term of performances.

3.2.5 IMPLEMENTING THE KIWI DRIVE

The kiwi drive is the way an omnidirectional robot with three omniwheels is being controlled in order to move in any direction without rotating (unless the robot is commanded to do so). With no template, we developed our own system: according to the angle the robot was supposed to move, we had to find the best power repartition in the three motors to ensure the right move and figure out how and when to stop it. Keeping the robot's orientation has always been a challenge all along the project and a necessity: when a robot or the drone changes its orientation, the system must know this change in order to modify accordingly the commands. We used different techniques in order to settle this matter

[19] *http://www.ludep.com/ground-unit-construction-and-structure-ideas-implementing-omniwheels/*
[20] *http://www.ludep.com/improving-the-omnidirectional-robot-design/*
[21] *http://www.ludep.com/arrival-of-the-new-omniwheels/*
[22] Rotacaster website : *http://www.rotacaster.com.au/*

but unfortunately, our environment did not allow us to make it work properly. This problem could have been fixed if the image analysis was able to detect oriented shapes instead of colors: we were about to implement it, it was just a matter of time and camera resolution.

The implementation of the kiwi drive has indeed evolved, from sending basic command lines for motors to the model we are using today: either a Xbox 360 controller (for the leader) or an autonomous driving mode run by the computer. The following articles show more details about this progression:

- *CONTROLLING THE LEGO BRICK VIA COMPUTER: THE BEGINNINGS OF THE KIWI DRIVE*[23]: this article shows the very first working tests with the communication between a brick and the computer via Bluetooth. It explains the basics of the communication layer, how the client and server exchange information, and commands.
- *WORKING OUT THE OMNIDIRECTIONAL BEHAVIOR OF THE KIWI DRIVE*[24]: you can find here the first model implemented for the control of a robot with the kiwi drive. It explains how we figured out the repartition of the powers, in every case. This is a rather empirical model that has been proved to be correct in the next articles when we came up with our complete mathematical model.
- *IMPROVING THE KIWI DRIVE: TESTS WITH A COMPASS SENSOR*[25]: the implementation of the compass sensor. The purpose, as previously mentioned, was to keep the robot's orientation all along the experiment. You will see how and why this could not be used for the sake of the project.
- *SOME STEPS FURTHER WITH THE OMNIDIRECTIONAL ROBOT*[26]: this is the final article we had for the omnidirectional implementation (and the one we are using that far). The mathematical model is really complete, its implementation is fully explained and you will know more about how we got rid of the rotation.

3.3 FLOCK BEHAVIOR

3.3.1 HARDWARE

The flock is being monitored with a computer running:

- OS - *Windows 7 32-bits*
- Processor - *Intel(R) Pentium(R) CPU 3,00GHz 2,99GHz*
- RAM - *2,00 GB*

We could indeed have chosen Linux in order to match the other application and be able to run everything on the same computer but we wanted to launch the flock handler from any computer from the university. This is not a serious matter because if needed the code would be easily transportable.

[23] *http://www.ludep.com/controlling-the-lego-brick-via-computer-the-beginnings-of-the-kiwi-drive/*
[24] *http://www.ludep.com/working-out-the-omnidirectional-behavior-of-the-kiwi-drive/*
[25] *http://www.ludep.com/improving-the-kiwi-drive-tests-with-a-compass-sensor/*
[26] *http://www.ludep.com/some-steps-further-with-the-omnidirectional-robot/*

3.3.2 SOFTWARE

We decided to use Java because of the closeness with LeJOS in order to have a "full Java layer" for the entire flock. We developed our last project under the same configuration and thought it was a good reason to keep it going that way (even if other languages seemed appealing, like Ruby for instance).

3.3.3 OUR IDEAS AND GOALS OF THE FLOCK

With all the projects we have reviewed, all the ideas we came out with, we had a really accurate template of flock we wanted to create. The most important features would be:

- **A dynamic flock:** the program has to know the units it controls in real time, check them if they are still in sight, if they are still alive or even close to die (low battery level for instance). The flock should correct its own formation while running according to the number of robots in the flock.
- **An oriented flock:** the units following the leader should consider its orientation and compute their positions accordingly. Whatever the direction the leader will take, the units should stay behind the direction the leader is facing as in the Figure 9 and Figure 10.

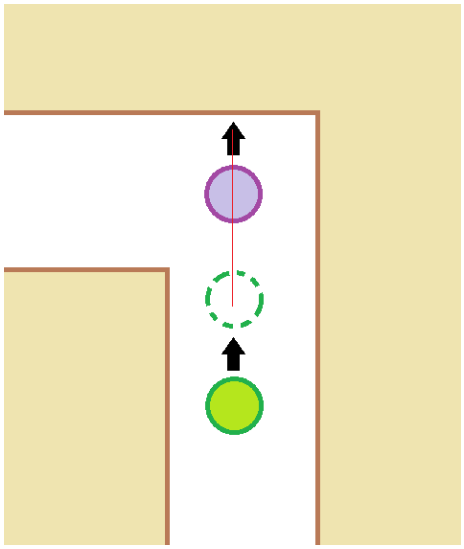


FIGURE 9: LEADER IN PURPLE, GREEN FOLLOWING

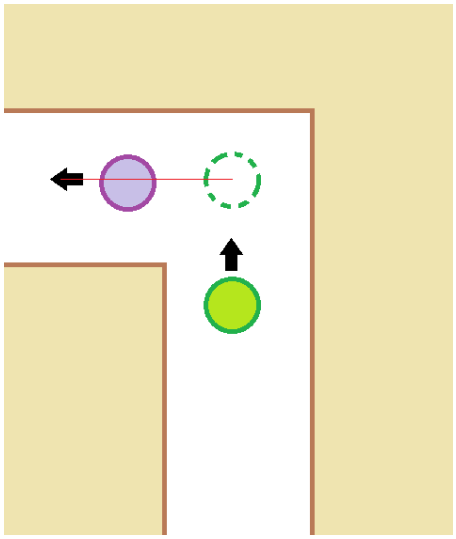


FIGURE 10: PURPLE CHANGES DIRECTION, GREEN STAYS "BEHIND" INSTEAD OF STAYING "BELOW"

- **A management of dead units:** this is a feature useful for the dynamic flock requirement. The program should know exactly when to put a unit aside, when to reintegrate it in the flock again.
- **A switchable leader:** a leader is a unit like another one and if dead, we need to be able to control another unit in order to lead the flock. The program should be able to do it when required or whenever the user wants to switch it.
- **An avoidance layer:** whenever a unit has to move, it checks before whether the move is authorized or not:
    - Is there another unit is on the way or not (c.f. Figure 11)?
    - Is there another unit that will cross its own movement?
    - Is it blocking the leader?
    - Is it about to go out of the field of view of the camera?

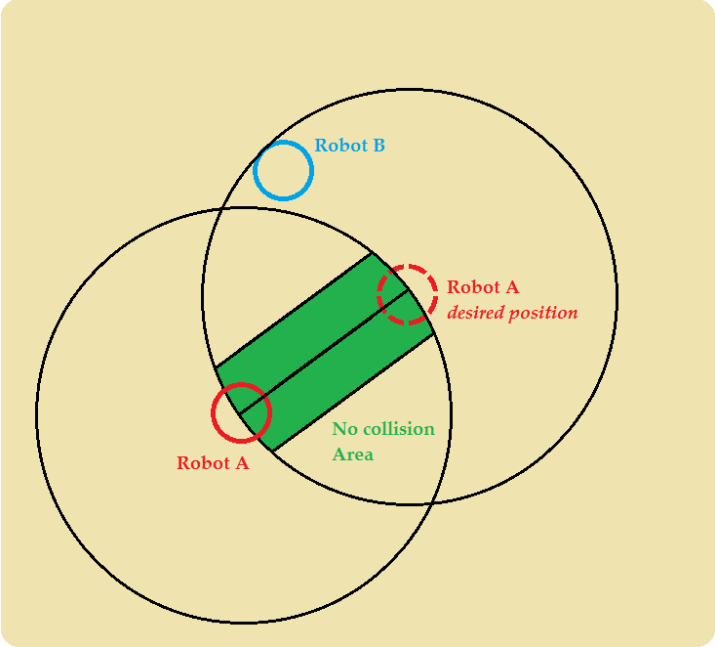


FIGURE 11: ROBOT A CHECKS IF ROBOT B WILL BE AN OBSTACLE DURING ITS MOVEMENT

Those are all the features that our system had to implement in order to define the flock behavior we wanted to design. The plan being developed, the next section will explain how we implemented it.

3.3.4 IMPLEMENTING THE FLOCK, STEP BY STEP

The flock behavior was really interesting to design because we did not need to create the whole program at once. Every feature we defined in the previous section could be added one by one and therefore improve our system every time adding new layers (c.f. Figure 12), upgrading it close to our expectations.

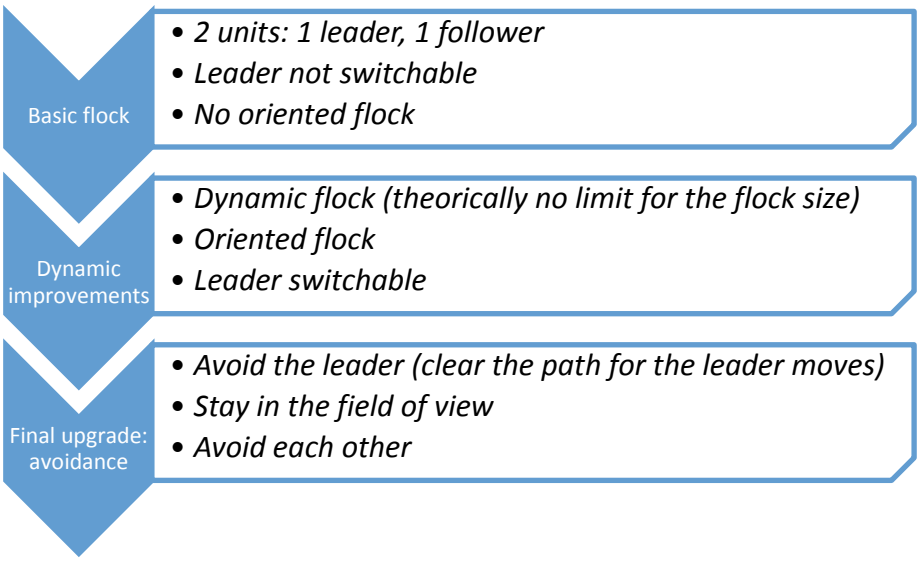


FIGURE 12: FLOCK BEHAVIOR SCHEME IMPLEMENTATION WITHIN THE PROJECT

In order to go further about the way it was implemented, coded, designed, those following articles should shed some light on the obscure points:

- *MERGING OUR WORK TOGETHER: THE BEGINNINGS OF WORTHY AND NOTABLE RESULTS*[27]: this article is the one we published just after connecting our two parts (drone and flock). All the features coded for the flock are clarified, the way to control the flock is being totally exposed and our first results and the concluding ones are explained even with videos;
- *THE FLOCK BEHAVIOR: FROM SCRATCH TILL NOW*[28]: the one and only article containing everything that is about the flock, that has been coded, that should be coded (if someone wanted to continue this project). We analyze our choices, how we settled the major problems and the outcome of them, with all the tests required and their explanations in order to show how our system responds in its first final version.

3.4 CONCLUSION

When we stepped back and took a closer look at the time we spent on the robots and the time on the flock, we realized that the robots design and implementation was twice longer than the flock one. To be honest, we thought it would have been the complete opposite. Nonetheless, after all, when we can think of it now with some hindsight, every single moment spent on the omnidirectional robots was worth it. The strength of the flock depends first on the reliability of its own units: how accurate they can be in their movements, how solid, fast and light they can be. Those two parts composing the flock can be dissociated, we can upgrade one without affecting this other (and reversely) and that gave once more our project more flexibility.

At the end, our flock is really different from what we know when we think about flocks: instead of having every unit choosing its own path, a hive-mind conducts everything and take care of every possible problem. Our system has its strengths as we demonstrate it before but it has indeed its weakness like every system. The biggest and most obvious one would be the crash of the hive mind: the flock would be totally lost and could not handle its own function. However, we did everything to make it safer, more reliable and handle all the fatal scenarios that could occur. The only event that could stop our system would be a power shortage; unless we launch our application from a laptop, a smartphone (which was our final objective) or anything that can run our code and has a battery.

[27] *http://www.ludep.com/merging-our-work-together-the-beginnings-of-worthy-and-notable-results/*
[28] *http://www.ludep.com/the-flock-behavior-from-scratch-till-now/*

4 MERGING EVERYTHING

The project has been so far separated into two distinct parts that are meant to become a unique, consistent system: the drone and the flock of omnidirectional robots. Since they were considered on their own, it is required to define a communication process in order to coordinate their actions.

This chapter will first focus on how communication between the drone and the robots is achieved and, on a larger picture, how all the data are transmitted between the different devices (drone, robots, and computers).

Results obtained through this merging and their conformity with the expected goals of the project are scrutinized in a second part of this chapter.

Eventually, a discussion is started on the possible improvements that could be brought to the system, as regards to the results.

4.1 COMMUNICATION PROTOCOLS

The biggest issue in the merging of our work lies in the communication between all our devices, and especially between the two parts we have explained so far in the previous chapter: the drone and the flock of robots. This chapter focuses on how we managed to put everything together, so as to have a working data exchange between the drone and the flock.

4.1.1 OVERVIEW

A proper communication service between all of the entities that are part of our system is a necessary condition to perform our tasks with good results. Basically, we need to achieve a goal really simple to present: our drone has to communicate data (e.g. a robot coordinates) with the leader of the flock of robots, which itself dispatches orders to the remaining of the flock. Then, from time to time, the leader passes its role to another robot, and the drone needs to get that information as well. If one of those communication lines is broken or not efficient, it is the whole system that risk failing in keeping a consistent behavior.

Since our ARDrone is WiFi enabled only, and our LEGO robots communicate with Bluetooth (we exclude USB in both cases, since it would not work for a mobile and independent system), we need an intermediate layer, that we call the Main frame. This layer has been split in two parts, with one specialized in communication with the drone, and the other with the flock. Separation was made so:

- We can work fast and separately on two different computers at the same time (remember we are two people in charge of this project), while running our tests and experiments independently. One station –PC1- deals only with the drone, and the other –PC2- with the flock.
- We can deal without lots of hassle with two different coding languages that are C/C++ and Java. C++ is required to use the official API for the drone and preferred for the image analysis, while Java is handy when it comes to deal with the LeJOS API embedded in the robots.

Therefore a third type of data exchange protocol is required, to transmit data within our Main frame to a station or another. This is handled by a socket connection that is managed by a regular client/server protocol. The upside of this solution is that **it is still possible to launch both of our programs on the same computer** without having to change anything apart from the IP address of the server.

Add to this system the possibility for users to interact with our programs by using controllers (we used both wired and wireless Xbox 360 gamepads), and you get a rather broad range of communication services (summarized in Figure 1) that need to be handled. This chapter provides more details about how the protocols have been established. The topic of the user input will not be mentioned, since it has already been explained in previous articles.

I.

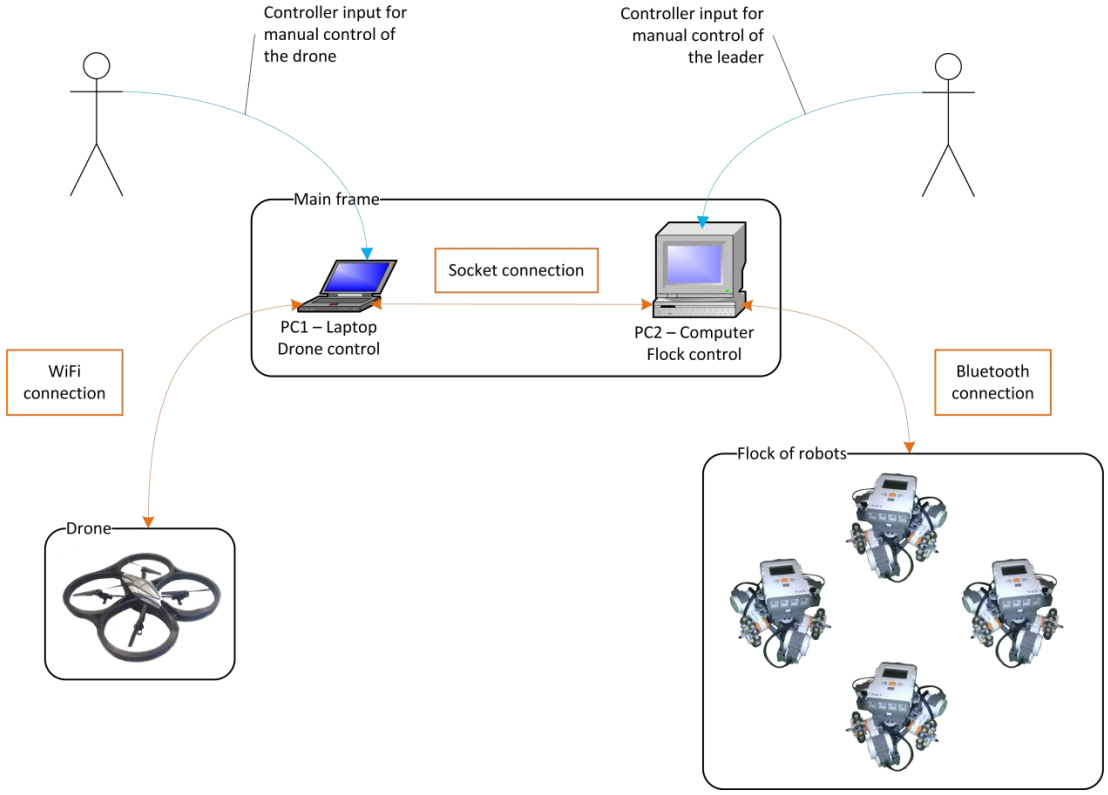


FIGURE 13: OVERVIEW OF THE EXCHANGE LAYERS IN OUR SYSTEM

4.1.2 DRONE WITH PC1: WIFI

The ARDrone's firmware was still locked by its developers at the time we write this report (rewriting our own embedded software is impossible), so all the programming that handles the behavior of the drone had to be done remotely, on the laptop we named PC1. We chose a laptop instead of a regular computer, insofar as it was the easiest way to test the drone in different locations – and also because computers of our university are not WiFi-enabled by default.

To put it in a nutshell, the drone sends navigation data (`navdata`) and a video stream to PC1, which reads them, runs its algorithm analyzing those data and then send commands to the drone, to tell it how to react to those information. Navdata contains all information about the drone's status, like its speed, its position (tilt and accelerometer sensors), its engine rotation speed, etc.

4.1.2.1 CONNECTING TO THE DRONE

The ARDrone behaves as a WiFi hotspot, to which every device can be automatically connected in ad-hoc mode. So before starting our program on PC1, you just have to use your usual WiFi connection manager on your OS to connect to the drone (for this to work, the PC needs to be the only device connected to it). Once this is done, exchange of different kinds of data may start.

Detailed information concerning the Wifi network and connection, and the communication services between the ARDrone and a client device, are provided in the end of the first of our appendices.

4.1.2.2 COMMUNICATION SERVICES

The whole interface between the drone embedded firmware and our own program is handled by the ARDrone library that belong to the official API. More specifically, it's the `ARDroneTool` library part of this library that implements the four communication services required for a proper connection: controlling and configuring the drone, receiving navdata, receiving the video stream, transferring critical data. The `ARDroneTool` library provides implementation tools to manage those services[29]:

- a command management thread, which collects commands sent by all the other threads, and send them in an ordered manner.
- a `navdata` management thread which automatically receives the navdata stream, decodes it, and provides the client application with ready-to-use navigation data through a call back function.
- a video management thread, which automatically receives the video stream and provides the client application with ready-to-use video data through a call back function.
- a `control` thread which handles requests from other threads for sending reliable commands from the drone, and automatically checks for the drone acknowledgements.

When created, those threads initiate the connection with the drone by themselves, and use another library (vp_com) to reconnect automatically to the drone if the connection is lost.

This chapter does not dwell on all the implementation details of this library, since they are already explained well in part 5.2 of the official *ARDrone SDK Developer Guide*.

4.1.3 PC2 WITH FLOCK: BLUETOOTH

The connection between the computer and the robots in the flock is handled by Bluetooth. We will expose how a client (robot) and the server (computer) interact with each other and how the connection is being created and maintained.

4.1.3.1 ON THE COMPUTER AND THE BRICK SIDE

The brick and the computer use one common tool (which is almost coded the same way and has been changed for convenient reasons) that we called `MessageFramework`. That class is supposed to:

- Connect one device to the other (brick ➔ computer or computer ➔ brick according to the device you are considering);
- Create the input and output streams (in order to send or receive messages later);

[29] As stated in part 3.3 of the official *ARDrone Developer Guide*.

- Have methods in order to send and treat incoming messages (a daemon is waiting messages coming from the brick) using the `LIMessage` class and making sure that:
    - Messages have flag showing the beginning and the end (ensuring that messages would come without any loss);
    - Messages would carry a type relevant to their content (basically, it would be `command` or `debug`) in order to treat them accordingly.

4.1.3.2 THE COMPUTER (/THE SERVER)

The computer is in charge of the connection with all the units at the same time. All the bricks are represented in the program by an instance of a class *Brick* pointing itself towards an instance of a `MessageFramework`. It is the computer version of `MessageFramework` that initiates the connection: the client is waiting for the server to attempt a connection. Every time a connection is lost, the server is closing its own connection with the client and is ready to create a new one (as soon as the client is waiting for the server). That way, the program does not need to restart when a client disconnects for any reason and the clients list is updated: it gave the system more flexibility. The server should be considered like the hive-mind of the flock. It is in charge of giving every order to the flock and it is really important for the server to know everything at any time: knowing the number of units connected/present in the flock is one of those parameters and it was useful at that point to know when a client was disconnected for instance.

4.1.3.3 THE BRICK (/THE CLIENT)

The brick waits for the server to open a connection. When it is done, the brick sends a heartbeat every half second in order to give feedback to the server in real time (we use this heartbeat to transmit the battery level of the brick for example; relevantly used, it can be used to set the dead units aside). If the server misses five of those heartbeats, the connection is lost and the brick considered dead (until it reconnects itself for some reason).

4.1.3.4 THE MESSAGES

Basically, the messages are string containing commands that could be interpreted in a shell. Every string received (either on the client or server side) is being split at every space. We iterate in that list of string and if a string matches a predefined command, we check if the numbers and the types of argument given after that are correct and, if so, we activate the proper method accordingly. This technique allows us to send several commands in the same string and therefore, trigger them at the same time.

4.1.4 PC1 WITH PC2: SOCKET

We built a client/server communication layer between PC1 and PC2. Figure 14 illustrates the typical exchange that constantly happens in real-time. PC1 - the client-, transmits data resulting from image analysis to PC2 - the server-, so that the leader can take decisions about the flock formation. This information is usually only valid at the time it is sent; a few seconds -or even a split second- later, it is not correct anymore, and is replaced by a new one. As for PC2, it only needs to send the id of the current leader when it changes. The data flow is therefore constant and almost continuous from the client to the server, whereas it is much sparser the other way around.

We agreed on our own custom communication protocol, where we exchange arrays of characters (strings of data) that are then parsed locally. This is a low level way of dealing with information transfer, and it overcomes well the fact that we use different languages on two machines.

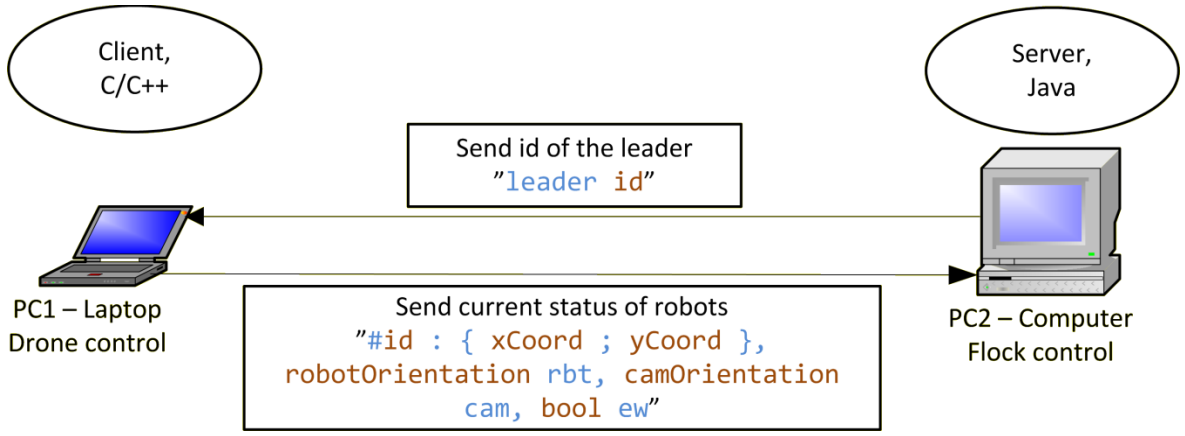


FIGURE 14: CLIENT/SERVER COMMUNICATION PROTOCOL ON A SOCKET. STRINGS TRANSFERRED RESPECT A PREDEFINED FORMAT. THE BLUE PART IS CONSTANT WHILE THE BROWN ONE CONTAINS THE ACTUAL INTEGER VALUES.

4.1.4.1 ESSAGES

Messages are represented by strings, which stand in of two different formats:

- From flock to drone:
  - `leader id`
  - Whenever a change of leader has been triggered, PC2 sends this information to PC1, so as to the drone changes its position accordingly.
  - Example: `leader 2` means that the new leader is the robot whose id is 2.
- From drone to flock:
  - `#id : { xCoord ; yCoord }, robotOrientation rbt, camOrientation cam, bool ew`
  - Whenever the image analysis algorithm goes through one iteration of its main loop, if a robot has been detected in a different location than the previous one, then PC1 sends this string to PC2. It enables a real-time update of the situation of the flock, in a way that does not over flood the bandwidth, since we only send the necessary data once, with an adjustable threshold to distinguish between two different locations.
  - Example: `#1 : { 1034 ; 207 }, 42 rbt, 172 cam, false ew` means that's robot 1 moved to the point of coordinates {1034, 207} with a 42° angle in respect to the drone orientation, while the drone was oriented in a 172° angle (0° being the North), and the robot did not trigger an edge warning (i.e. it is not close to being out of the camera field of view).

4.1.4.2 STARTING THE CONNECTION

The IP address of the server (PC2) is required on the client side, while they both have to agree on a port where they open the socket. When the connection starts, both programs on PC1 and PC2 start a thread that runs continuously to listen to events on the same socket. When a new message is received on a device, the program parses it and triggers the corresponding event.

4.1.4.2.1 CLIENT SIDE

The client is a regular C client that we coded and embedded into a C++ `Client` class. This class is instantiated once the user clicks on the appropriate button in the GUI on PC1. This GUI is created by the `ColorMatcher` class, which can run by itself outside the drone program (meaning that we may also use a webcam or any video stream instead of one of the drone embedded cameras). Two really simple functions, at the `ColorMatcher` level, handle the client and illustrate its initialization:

```
void ColorMatcher::startConnections() {
        if(NULL == client) client = new Client(IPHOST, PORTNO);
        CONNECTION = 1;
}

void ColorMatcher::stopConnection() {
        delete client;
        CONNECTION = 0;
}
```

At a higher level, in the main program of the drone, happing in the `Planner` class, this connection handling is fully transparent. Once `ColorMatcher` is instantiated, `Planner` has access to the last registered leader number through an accessor. Figure 15 summarizes those different levels.

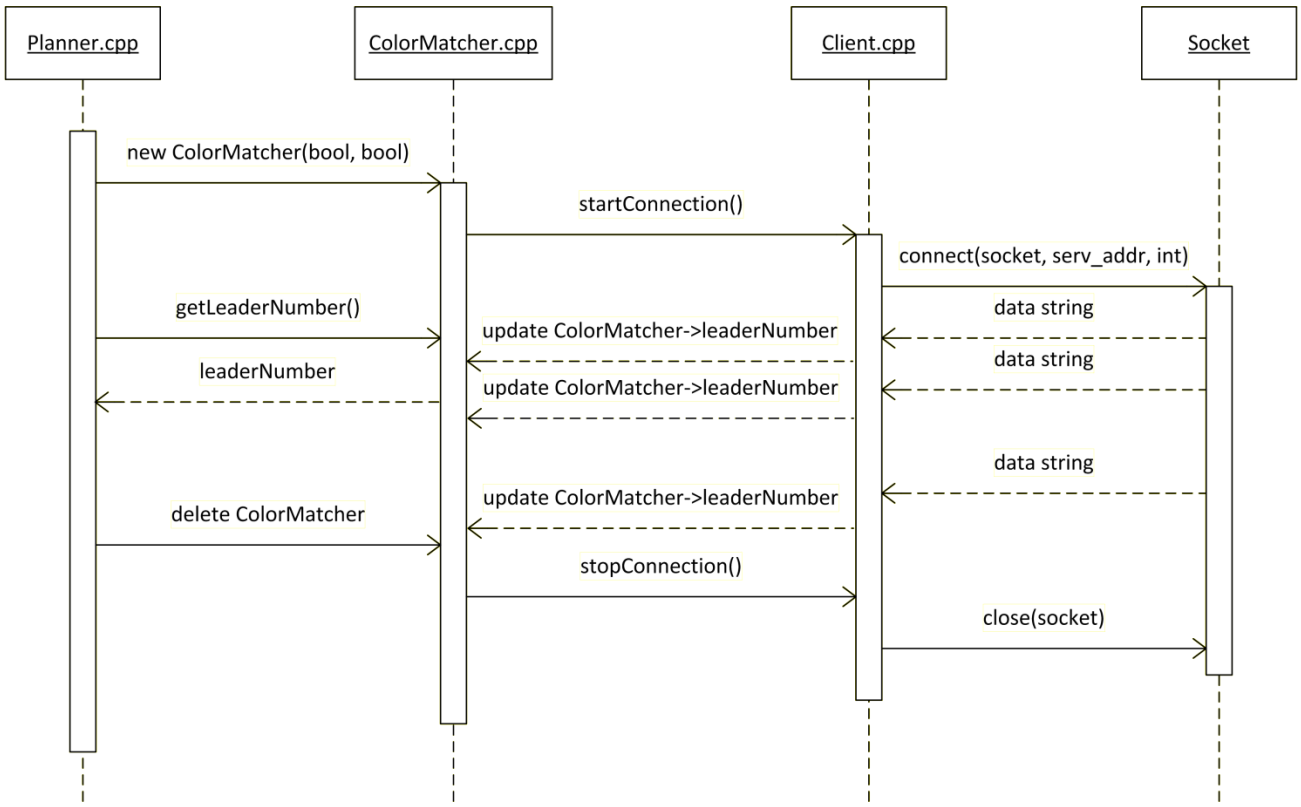


FIGURE 15: SEQUENCE DIAGRAM ILLUSTRATING THE LEVELS IN THE COMMUNICATION PROCESS ON PC1 (DRONE'S SIDE)

4.1.4.2.2 SERVER SIDE

The server is a basic Java program designed to be waiting all the time for the client to connect. We just have one client in our system, this was not tricky to handle and this allowed us to have a server that does not need to restart when the connection with the client is lost. That way, once the server was initiated, it would be connected all the time to the bricks, regardless to the status of the client.

Basically, the idea is in the following pseudo-code:

```
while ( not connected to client )
{
            open socket and wait for client to connect

            // client connected
            while ( messages received are not null && client is connected )
            {
                        treat messages
            }

            // client disconnected
            close socket
}
```

When there is no connection, the server waits for the client to connect. When it happens, the server accepts the connection, starts a daemon in order to treat the messages. As soon as the connection is stopped for any reason, the server stops the daemon, closes the connection and iterates the same routine described above.

4.2 RESULTS

Working on two different technologies for six months was interesting because we could look at each other code, give advices with hindsight and take the time to think of problems together with lateral thinking. Around one month before the end of the project, we had to experience a whole new setup: putting our work together and make it work properly. Making the integration did not take long (just a matter of few hours), but making the whole system work as fast as it needed to be (real-time behavior expected), adding features in order to make it more consistent, testing and tuning all the parameters so as to get the best results made those last weeks really busy and were worth our time.

4.2.1 OUR EXPECTATIONS

The best and most honest way to expose our expectations would be to show our description of our project. We wrote it before the implementation and it describes how we thought of the project six months ago:

"An unmanned aerial vehicle (UAV) is keeping track of a ground robot that leads a flock of similar robots. The UAV flies autonomously and uses its cameras to help the units moving in a defined formation, while communicating with the leader through an interfacing device."

The coloration splits the project in its main objectives. Namely, we wanted:

- ♦ The drone making image analysis using its camera and being capable to locate as accurately as it could be the robots on the ground;
- ♦ A flock led by one *switchable* leader;
- ♦ The drone to be autonomous, stay above the leader and move as smooth as possible in order not to cause problems for the image analysis;
- ♦ A dynamic formation that could handle itself in real-time, no matter what could happen;
- ♦ A centralized platform for all the several programs to run and take care of all the connections.

4.2.2 AFTER MERGING OUR PARTS

After we merged our two separated work, we were able to upgrade our own parts according to the results we obtained during the testing period. The following articles describe how each of our parts evolved after we put all of our code together:

- *THE FLOCK BEHAVIOR: FROM SCRATCH TILL NOW*[30]: all the flock implementation and improvements have actually been done after we merged our work. We could not have done any tests without the drone, therefore you can read in this article how the flock has been implemented step by step.
- *FINAL WORDS ON THE DRONE: MERGING PREVIOUS WORK WITH NEW TASKS*[31]: this article is particularly relevant if you want to know the last upgrades made for the drone flight features, namely hovering, how to stay above the leader autonomously and switching leader in real-time.

4.2.3 THE PROJECT IN ITS FIRST FINAL VERSION

We will make a comparison between what we expected to have and what we really have today. That way, we can have a fair and honest list of everything that *has* been done and *should* have been done. What we *could* have done will be treated in the next section (*c.f. Improvements*) in order to be coherent and show exactly what our project responds to at this very moment.

This next table takes our initial expectations as standards. The system we have today has encountered lots of improvements and we have more recent needs that will be explained in the next section, as previously mentioned.

[30] *http://www.ludep.com/the-flock-behavior-from-scratch-till-now/*
[31] *http://www.ludep.com/final-words-on-the-drone-merging-previous-work-with-new-tasks/*

	Implemented features	Extra features not implemented
Drone	Detect dynamic number of units on the ground; Autonomously follow the leader; Capable of switching leader; Move smoothly for image analysis performances.	Detect dynamically, on the go, more units than initially planned in the compiled code; Detect orientation of the ground units (hardware related issue).
Flock	Have a solid platform for hosting clients (drone and ground units); Dynamic flock (that handles its formation in real-time) taking care of unit status (death, loss…); Switchable leader; Avoidance issues.	Robot orientation (coded and implemented but environment related issue).
Overall	Reliable response time (reducing the delays has been a bullet point for this project); Portable system;	Centralized program on a smartphone (we can already have all our code on the same machine)

However all those extra features represent minor problems, meaning that they are not essential for the program to work; but it would have surely added a nice value to have them coded if we had the time for it. Indeed our program works properly now and having those features would be adding another layer on our project.

4.3 IMPROVEMENTS

The next features we will expose you are not part of our initial "contract": the project's goals have been previously mentioned and reviewed. This following part deals with all the ideas we came out with during the project and that could be worth the implementation. If we had the time, those ideas would be our next goals: it would without a doubt improve the project in a significant way.

4.3.1 ABOUT THE FLOCK

- If the camera cannot detect the orientation of the robots, they need to stabilize their own orientation by themselves. We already tried with a compass and this can be source of many problems due to magnetic disturbances. The use of an accelerometer coupled with an gyroscope could be an alternative to the correction we do according to the wheels position – and bring correction and accuracy to our dead-reckoning;
- Every robot could be given more responsibility. For the time being, a robot simply sends information about its status (battery level for instance) but does not choose its own action: they do not have any autonomous behavior. We could add some more sensors, think of an additional avoidance layer or any other autonomous behavior. The only problem would be to respect the hive-mind's (server) authority because it is in charge of the whole flock. This could be the starting point for the implementation of the subsumption architecture.
- Give an autonomous behavior to the leader and make the whole system out of human control;
- Make an automated change/selection of leader according to the unit properties, position or status.

4.3.2 ABOUT THE DRONE

- Go from a PID controller to a non-linear stabilization algorithm specially adapted to the quadrotors design. It may however be impossible to achieve on the AR.Drone as long as developers do not have access to its firmware. Yet, it would improve its stability while tracking a tag without a doubt.
- Become less light-dependent in terms of tag detection. Shape detection instead of color-based detection should be an easy first step, providing that the low resolution does not prevent it when it has to deal with multiple robots in the field of view. Experimenting with LEDs embedded on top of each robot and blinking at different frequencies depending on the robot might be a huge step forwards to a more reliable system.
- Dealing with the short battery-life of the AR.Drone, by planning shifts among a set of drones (one flies, while others wait to replace it at the right time).
- Implementing obstacles detection with the image analysis and sending coordinates about them to the leader. This would actually be quite easy to implement in a controlled environment with tagged obstacles (since we already have everything needed for that), yet much harder in an unknown setup (which remains the goal of the system).
- Trying of flock of drones which would try to optimize the coverage of the ground and keep together track of as much ground units as possible.

4.3.3 ABOUT THE WHOLE SYSTEM

- Detecting the orientation of the robots would totally suppress the need to have a reference point. The system would be totally working by itself without any help from the outside. Even if the robots would change their orientations (for any reason), the system would adapt and make them move accordingly simply by changing the command or asking the robots to point towards the same direction than the drone for example;
- Add a “rescue mode”: every lost unit should have a last position and the drone could attempt to find a lost unit and bring it back to the flock. The flock could wait in the meanwhile or even follow the drone (the drone could even become the leader for this purpose);
- Put a GPS chip on the drone and send it on a reckon mission with the flock below it, to have them reach a defined GPS location together.

5 CONCLUSION

All the goals we set in this initial challenge that defined our project are totally fulfilled today. The project will always ask for improvements but the version we delivered is fully working and gave us a reliable proof of concept. All the upgrades we suggested could be the starting point of another project: there are so many different ideas to explore, concepts to develop, that we wish we had some more time to investigate them.

If you were watching our project running on its first final version, you would see two computers working together in order to make a drone fly over a flock. The drone would be manually controlled or follow autonomously the leader, the flock would dynamically adapt itself in real-time. For the time being, the leader of the flock is remote-controlled as we wanted it to be, but the next step would be to make it autonomous in order to have the whole system out of human control. Communication layers have been long and deeply studied in order to give the real-time experience we have when testing our project. The system showed its efficiency and could evolve in so many different ways, namely military (exploration or rescue missions), entertainment (games, exhibitions), education and even research (upgrading the way the flock is being handled, adding behaviors would be a restless process).

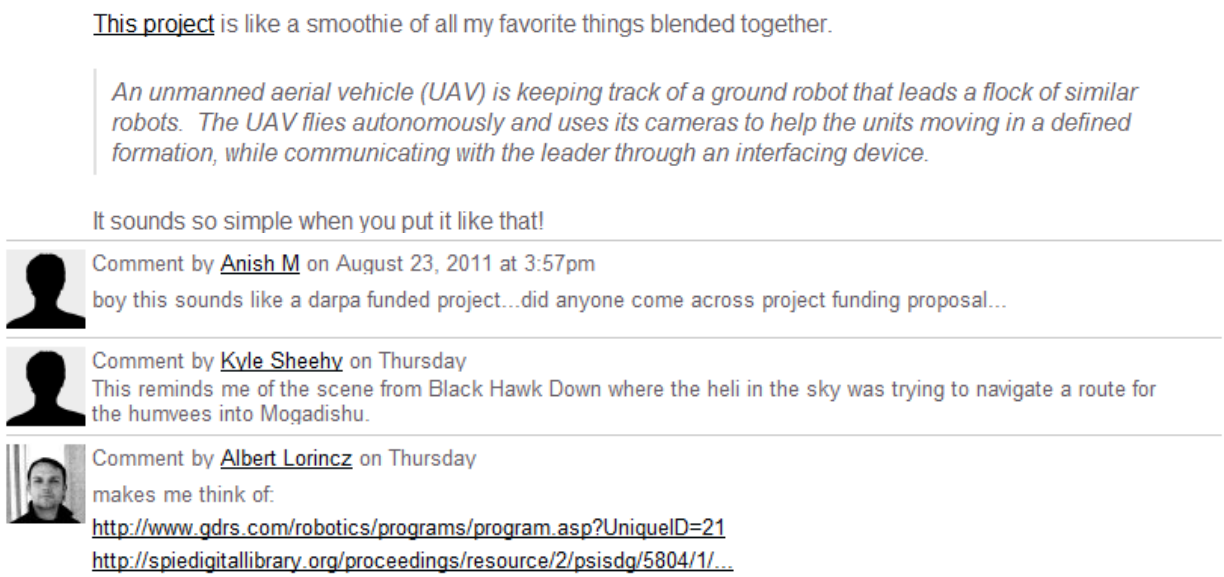
We wanted to present an open-source, easy to set up, portable project and this is what we have so far. We are really satisfied to know that our website gives fuel for thoughts to people that are or might be intrigued by different cutting-edge technologies and merging them together. As an end note for this conclusion, we selected some of the comments and appreciations we have been quoted in within the time we have been doing this project and proudly display them on the next page.

PARROT AR.DRONE REFERENCING OUR PROJECT IN THEIR MICRO BLOGGING SERVICE[32]

MORFARS.DK LINKING OUR WEBPAGE THROUGH THEIR PERSONAL PAGE[33]

"When you are walking down the Hopper-0 corridor, you may hear a strange voice coming from Hopper-035. When you look into the office you find two young men in deep concentration. One is experimenting with a large four-circled drone flying around the office making a lot of noise and a lot of wind. The other is sitting on the floor with a LEGO robot car in his hand, trying to figure out the best way for it to drive and turn in any direction."

EXTRACT FROM THE ARTICLE POSTED IN INSIGHT@KATRINEBJERG[34]



POST AND COMMENTS TAKEN FROM DIY DRONES[35][36][37]

[32] *http://twitter.com/#!/ardrone*
[33] *https://www.facebook.com/morfars.dk*
[34] *http://katrinebjerg.au.dk/news/artikel/a-flying-drone-invades-hopper-0/*
[35] *http://diydrones.ning.com/profiles/blogs/ar-drone-android-lego-mindstorms-nxt-bliss*
[36] First link in the comments: General Dynamics *http://www.gdrs.com/robotics/programs/program.asp?UniqueID=21*
[37] Second link in the comments: SPIE *http://www.gdrs.com/robotics/programs/program.asp?UniqueID=21*

6 APPENDICES

6.1 APPENDIX A: AR.DRONE OVERVIEW

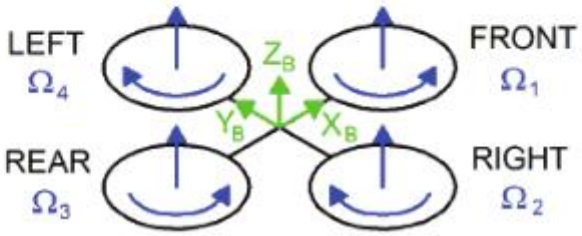
The following pages are extracted without modification from the official AR.Drone SDK Developer Guide (v1.7) PDF made by Parrot. They provide a good overview and technical details as well about the AR.Drone. This is a must read for the advised reader who wants to get a better understanding of the project.

Note: **UAV** stands for Unmanned Aerial Vehicle and refers to the AR.Drone itself.

2.1 Introduction to quadrotor UAV

AR.Drone is a quadrotor. The mechanical structure comprises four rotors attached to the four ends of a crossing to which the battery and the RF hardware are attached.

Each pair of opposite rotors is turning the same way. One pair is turning clockwise and the other anti-clockwise.



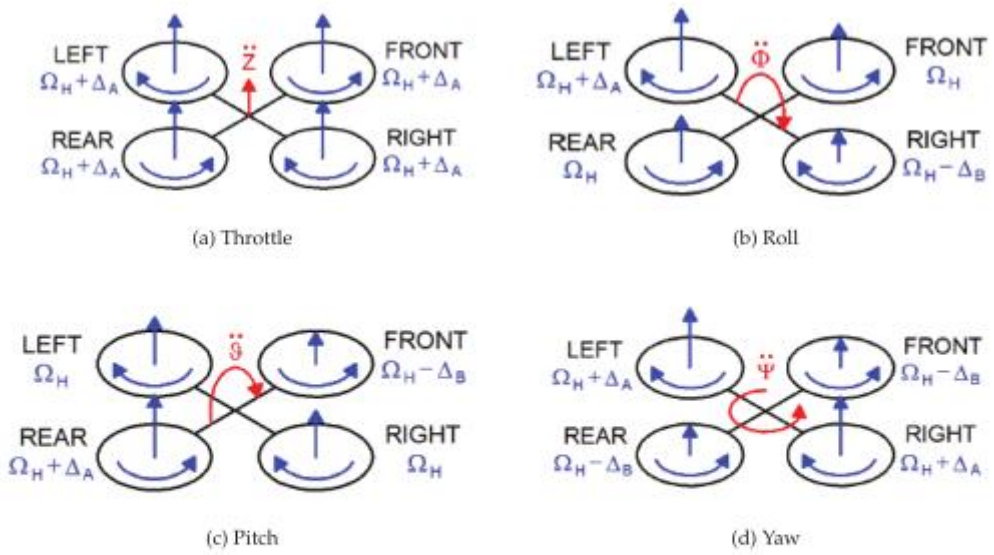
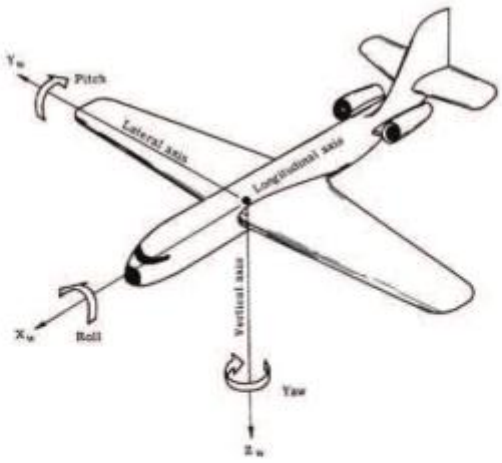


Figure 2.1: Drone movements

Manoeuvres are obtained by changing pitch, roll and yaw angles of the AR.Drone .



Varying left and right rotors speeds the opposite way yields roll movement. This allows to go forth and back.
Varying front and rear rotors speeds the opposite way yields pitch movement.
Varying each rotor pair speed the opposite way yields yaw movement. This allows turning left and right.

(a) Indoor (b) Outdoor

Figure 2.2: Drone hulls

2.2 Indoor and outdoor design configurations

When flying outdoor the AR.Drone can be set in a light and low wind drag configuration (2.2b). Flying indoor requires the drone to be protected by external bumpers (2.2a).

When flying indoor, tags can be added on the external hull to allow several drones to easily detect each others via their cameras.

2.3 Engines

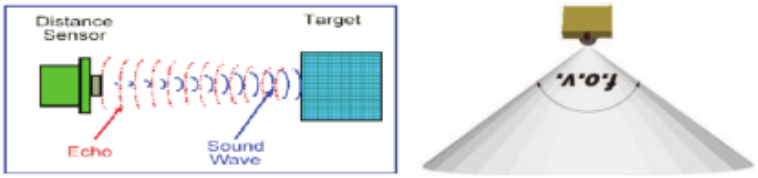
The AR.Drone is powered with brushless engines with three phases current controlled by a micro-controller

The AR.Drone automatically detects the type of engines that are plugged and automatically adjusts engine controls. The AR.Drone detects if all the engines are turning or are stopped. In case a rotating propeller encounters any obstacle, the AR.Drone detects if any of the propeller is blocked and in such case stops all engines immediately. This protection system prevents repeated shocks.

3839

[38] **Brushlessengines** are electric motors powered having electronic commutation systems, rather than mechanical commutators and brushes. The current-to-torque and frequency-to-speed relationships of brushless motors are linear. (cf. *http://en.wikipedia.org/wiki/Brushless_DC_electric_motor* & *http://en.wikipedia.org/wiki/Brushless_AC_electric_motor*)

[39] **Three-phase electric power** is a common method of alternating-current electric power generation, transmission, and distribution. (William D. Stevenson, *Jr. Elements of Power System Analysis Third Edition*, McGraw-Hill, New York (1975). ISBN 0070612854. Page 2)



(a) Ultrasound sensor (b) Camera

Figure 2.3: Drone Sensors

2.4 LiPo batteries

The AR.Drone uses a charged 1000mAh, 11.1V LiPo batteries to fly. While flying the battery voltage decreases from full charge (12.5 Volts) to low charge (9 Volts). The AR.Drone monitors battery voltage and converts this voltage into a battery life percentage (100% if battery is full, 0% if battery is low). When the drone detects a low battery voltage, it first sends a warning message to the user, then automatically lands. If the voltage reaches a critical level, the whole system is shut down to prevent any unexpected behaviour.

2.5 Motion sensors

The AR.Drone has many motions sensors. They are located below the central hull.

The AR.Drone features a 6 DOF, MEMS-based, miniaturized inertial measurement unit. It provides the software with pitch, roll and yaw measurements.

Inertial measurements are used for automatic pitch, roll and yaw stabilization and assisted tilting control. They are needed for generating realistic augmented reality effects.

An ultrasound telemeter provides with altitude measures for automatic altitude stabilization and assisted vertical speed control.

A camera aiming towards the ground provides with ground speed measures for automatic hovering and trimming.

2.6 Assisted control of basic manoeuvres

Usually quadrotor remote controls feature levers and trims for controlling UAV pitch, roll, yaw and throttle. Basic manoeuvres include take-off, trimming, hovering with constant altitude, and landing. It generally takes hours to a beginner and many UAV crashes before executing safely these basic manoeuvres.

Thanks to the AR.Drone onboard sensors take-off, hovering, trimming and landing are now completely automatic and all manoeuvres are completely assisted.

4041

[40] **DOF**: Degree of freedom

[41] **Microelectromechanical systems (MEMS)** (also written as micro-electro-mechanical, Micro Electro Mechanical or microelectronic and microelectromechanical systems) is the technology of very small mechanical devices driven by electricity. (cf. *http://en.wikipedia.org/wiki/Microelectromechanical_systems*)

User interface for basics controls on host can now be greatly simplified :

- When landed push *take-off* button to automatically start engines, take-off and hover at a pre-determined altitude.
- When flying push *landing* button to automatically land and stop engines.
- Press *turn left* button to turn the AR Drone automatically to the left at a predetermined speed. Otherwise the AR Drone automatically keeps the same orientation.
- Press *turn right* button to turn the AR Drone automatically to the right. Otherwise the AR Drone automatically keeps the same orientation.
- Push *up* button to go upward automatically at a predetermined speed. Otherwise the AR Drone automatically stays at the same altitude.
- Push *down* to go downward automatically at a predetermined speed. Otherwise the AR Drone automatically stays at the same altitude.

A number of flight control parameters can be tuned:

- altitude limit
- yaw speed limit
- vertical speed limit
- AR.Drone tilt angle limit
- host tilt angle limit

2.7 Advanced manoeuvres using host tilt sensors

Many hosts now include tilt motion sensors. Their output values can be sent to the AR.Drone as the AR.Drone tilting commands.

One *tilting* button on the host activates the sending of tilt sensor values to the AR.Drone . Otherwise hovering is a default command when the user does not input any manoeuvre command. This dramatically simplifies the AR.Drone control by the user.

The host tilt angle limit and trim parameters can be tuned.

2.8 Video streaming and tags detection

The frontal camera is a CMOS sensor with a 90 degrees angle lens.

The AR.Drone automatically encodes and streams the incoming images to the host device. QCIF and QVGA image resolutions are supported. The video stream frame rate is set to 15 Hz.

Tags painted on drones can be detected by the drone front camera. These tags can be used to detect other drones during multiplayer games, or to help a drone find its way in the environment. Both tags on the external and internal hull can be detected.

(a) 2D tags on outdoor shell (b) 2D tags on indoor shell

Figure 2.4: Drone shell tags

2.9 Wifi network and connection

The AR.Drone can be controlled from any client device supporting the Wifi ad-hoc mode. The following process is followed :

1. the AR.Drone creates a WIFI network with an ESSID usually called *adrone_xxx* and self allocates a free, odd IP address.
2. the user connects the client device to this ESSID network.
3. the client device requests an IP address from the drone DHCP server.
4. the AR.Drone DHCP server grants the client with an IP address which is :
   - the drone own IP address plus 1 (for drones prior to version 1.1.3)
   - the drone own IP address plus a number between 1 and 4 (starting from version 1.1.3)
5. the client device can start sending requests the AR.Drone IP address and its services ports.

The client can also initiate the Wifi ad-hoc network. If the drone detects an already-existing network with the SSID it intented to use, it joins the already-existing Wifi channel. 4243

[42] **CIF** (Common Intermediate Format), is a format used to standardize the horizontal and vertical resolutions in pixels of YCbCr sequences in video signals, commonly used in video teleconferencing systems. **QCIF**(Quarter CIF) offers a resolution of 176*144 pixels.

[43] **QVGA**(**Quarter Video Graphics Array**) is a popular term for a computer display with 320×240 display resolution.

2.10 Communication services between the AR.Drone and a client device

Controlling the AR.Drone is done through 4 main communication services.

Controlling and configuring the drone is done by sending *AT commands* on UDP port 5556. The transmission latency of the control commands is critical to the user experience. Those commands are to be sent on a regular basis (usually 30 times per second). The list of available commands and their syntax is discussed in chapter 6.

Information about the drone (like its status, its position, speed, engine rotation speed, etc.), called *navdata*, are sent by the drone to its client on UDP port 5554. These *navdata* also include tags detection information that can be used to create augmented reality games. They are sent approximatively 30 times per second.

A video stream is sent by the AR.Drone to the client device on port 5555. Images from this video stream can be decoded using the codec included in this SDK. Its encoding format is discussed in section 7.2.

A fourth communication channel, called *control port*, can be established on TCP port 5559 to transfer critical data, by opposition to the other data that can be lost with no dangerous effect. It is used to retrieve configuration data, and to acknowledge important information such as the sending of configuration information.

[44]

[44] **SDK**: Software Development Kit

6.2 APPENDIX B: HOW TO RUN THE PROJECT

Making our project work can be really tricky if you are not given any instruction. That is the reason this appendix will guide you step by step in order to make this project run with you own configuration. You will be told what the requirements are, how to deal with the drone, the computer (server and client) and the bricks and how to make sure to install the whole system properly.

We cannot guarantee you this will work but we followed these very instructions on other computers and it worked. We dealt with the major issues that you might encounter and solved them in order to make your time worth.

6.2.1 REQUIREMENTS

This project involves quite a lot of different pieces of hardware. Here is what we used in our setup. Please note that this a mere presentation of a working configuration and that it may probably vary a lot, especially on the computers side:

- An AR.Drone by Parrot, with firmware 1.5.1
- Four omniwheels robots (see the next Appendix for building instructions). More or less can be used (for more, one line of code per robot has to be added in the code on the drone side – in `ColorMatcher.cpp`), with a color tag on the top. They are using each an NXT brick running LeJOS 0.8.5 beta, with BlueTooth enabled.
- One Wi-Fi enabled computer running Linux, Ubuntu 10 or later. It handles the communication with the drone.
- One Bluetooth enabled computer with a Windows 7 OS. It manages the flock of robots.
- Both computers should be on the same network, with distinct IP addresses (first one is the client, second one the server). It should also be possible to run both programs on the same computer (at least first with a virtual machine because of the different OS), without too much change.

6.2.2 DOWNLOAD THE ARCHIVE

The code for the whole project (drone & robots), with all the required libraries and APIs, as it is at the time we write this report, is in our download section available here:
http://code.google.com/p/ludep/downloads/list.

It is included in one archive: `LUDEP_complete_archive.zip`.

To make sure you get the last version of the code, it may be easier to check it out directly on the SVN with your local repository manager (You may however not get all the necessary libraries for the robots):

```
svn checkout http://ludep.googlecode.com/svn/trunk/ ludep-read-only
```

6.2.3 DRONE SETUP

The following is an extract from the `Quickstart.txt` file that is in the folder `Drone_app_code_sdk_1.7/`
It is an adapted version of Cooper's Instructions[45].

[45] *csb88@cornell.edu*

6.2.3.1 OVERVIEW:

Project tested with **Linux Ubuntu 11.04** Natty Narwhal (and also previously with Ubuntu 10).

For most applications, only the planner.cpp file needs to be edited.

You must have your computer's Wi-Fi connected to the drone before running the program.

It is here advised to rather use firmware `1.5.1` (in folder `./drone_firmware_1.5.1`) which was the one used during development, and API `1.6` (which is available on our repository) or later.

6.2.3.2 QUICK TIPS/DESCRIPTIONS:

- `./ARDrone_API` - the API for the ARDrone
- `./application` - our application/code, derived from their Linux example
- `./application/Build` - where to build/run our application
- `./application/Sources` - our source code
- http://projects.ardrone.org - support/documentation website

6.2.3.3 DEPENDENCIES

Known Dependencies (there might be more):

```
libhighgui-dev
g++
libsdl-dev
libfftw3-dev
libiw-dev
```

On Ubuntu, the command '`sudo apt-get install g++ libsdl-dev libfftw3-dev libiw-dev libhighgui-dev`' should install everything you need for dependencies (including `opencv`).

6.2.3.4 TO BUILD:

1. Navigate to `./application/Build` in a terminal
2. run the command `'make'`
3. the last command displayed starts with `'cp'` if all goes well.

Undefined reference to `_main` or cannot find `-lpc_ardrone`?
Navigate to ./ARDrone_API/ARDroneLib/Soft/Build and run 'make'

6.2.3.5 TO RUN:

1. Connect your Wi-Fi to `ardrone_######` (drone must be on).
2. Once connected, in the Build directory, run '`./linux_sdk_demo`'

6.2.3.6 TO USE:

Currently, it is only configured and tested to work with the joystick/360 gamepad. See `./Documentation/x360_controller_drone.png` for control configuration. It should though also work with any other USB controller.

'Start/Stop custom algorithm' switches between manual flight and computer controlled flight (coded in `./application/Sources/UI/planner.cpp`).

The Graphical User Interface (GUI) is all about managing the configuration for image analysis, and real-time tuning. It is possible to start or stop the `IO` communication by socket (activated by default on port `4242`, on `localhost` -> this can be changed in:
`./application/Sources/ColorMatching/ColorMatcher.cpp`) on the GUI.

6.2.3.7 TO ADD NEW SOURCE FILES TO MAKE:

1. Open `'Makefile'`
2. add your source to the list under `'GENERIC_BINARIES_COMMON_SOURCE_FILES'`
3. headers do not need to be added to the `makefile`.

6.2.3.8 ADDING LIBRARIES/DEPENDENCIES:

1. Open `'Makefile'`
2. add your library to `'GENERIC_LIBS'`, with `-l prefix`.

6.2.3.9 MISSING A DEPENDENCY?

1. open System->Administration->Synaptic Package Manager
2. Search for the dependency
1. Note: install the 'dev' version of the libraries.
2. i.e. for `opencv/highgui: 'libhighgui-dev'` is the desired package
3. Right-click -> Mark for installation.
4. Click apply on the top.
5. Watch Ubuntu download and handle the dirty work.
6. Try compiling again.

6.2.3.10 FURTHER PROBLEMS WITH `MAKE` OR AT EXECUTION?

In the directory where you want to build your project:

1. run `‘make clean’`
2. run `‘make’` once more

6.2.3.11 USB WEBCAM-ONLY MODE

It is possible to test the color detection alone, while still being able to send data to a server, by using a USB webcam without even connecting to the drone. To achieve that:

1. Navigate to `./application/Sources/ColorMatching` in a terminal
2. run the command `'make'`
3. if all goes well, the last command printed in the terminal should start with `‘g++ -g –o ColorMatcher…’`
4. in the same current directory, run `‘./ColorMatcher’` (default webcam used will be the one plugged on `/video0`)

Communicating with a server is still possible.

6.2.3.12 STARTING A LOCAL *FAKE* SERVER

To get all the messages sent by the detection algorithm to the server, it is possible to simulate it by running a local server that listens to messages on the same port (default: 4242). In order to do that:

1. Navigate to `./application/Sources/ColorMatching/TestTools` in a terminal
2. run the command `'make'`
3. if all goes well, the last command printed in the terminal should be `'g++ -g –o Server server.o –pthread'`
4. in the same current directory, run `'./Server'`
5. Messages received will directly be printed in the current terminal

You may also want to test the client/server connection only. To do that:

1. Follow the previous instructions and start the server
2. In the same directory, run `'./Client'`
3. The client will just send one test message to the server

6.2.3.13 CHANGING THE IP OF THE SERVER

When the whole program is launched, or at least the image color detection, the GUI has a slider to toggle the connection to a remote server. This IP is hardcoded, and has to be changed in the code itself before building the program:

1. Navigate to `./application/Sources/ColorMatching/`
2. Open `ColorMatcher.cpp` with a text editor
3. Find the method `'void ColorMatcher::setDefaultValues()'` in the end of the file
4. Change the value of `IPHOST` to the IP where your server is running (if it is on the same machine, enter `'IPHOST = 127.0.0.1'`)
5. Run `make` again in the same directory

6.2.3.14 ANY MORE QUESTIONS?

Check our website for more information[46]; otherwise feel free to e-mail us: `michael@ludep.com`

6.2.4 FLOCK SETUP

This part is made for the setup of the brick and the computer in charge of the flock behavior. Note that we used Eclipse all along as our favorite IDE (Integrated Development Environment).

6.2.4.1 ON THE BRICK SIDE

This should be done quite easily. You need to send the `NXT.java` program on the brick, making sure you flashed your brick under LeJOS 0.8.5 beta firmware.

6.2.4.2 ON THE COMPUTER SIDE

The first thing you have to make sure is to have checked all the following points on this list:

- Have Windows as an OS (we had to be able to run our project on the university's computers, that is why we used Windows 7);
- Install LeJOS **0.8.5 beta** (we tried with the newest release and it didn't work);
- Install the libraries for the Xbox 360 Controller (`XboxControllerBase` folder in under Computer folder);
- Make sure the bricks' Bluetooth mode is activated on the bricks, and that they are visible (it will certainly help for the next step);

[46] *http://www.ludep.com*, with a full report in our *"Links & downloads"* page

- Add all the bricks through the Bluetooth device manager that you are going to use for the experiment and note their name and their MAC Address.

Once all of this is done, you have to modify some lines of code in order to make the program work with your own configuration. Basically, you will have to edit `Computer.java` in the Control in the package `Command`.

```
import java.awt.Point;

//! Class to run the program on the computer
public class Computer implements IServerCom {

    private ArrayList<Brick> brickList;
    private Brick brickInControl;
    private Controller cont;
    private MainGUI mg;
    private ServerCom sc;
    private FlockHandler fh;
    private int port = 4242;

    //! Constructor
    public Computer(){

        setBrickList(new ArrayList<Brick>());

        getBrickList().add(new Brick(1, this, "Branson", "0016530DB4A2"));
        getBrickList().add(new Brick(2, this, "Jambon", "0016530DB4FC"));
        getBrickList().add(new Brick(3, this, "Pampa", "0016530BEA38"));
        getBrickList().add(new Brick(4, this, "Poule", "00165312EFC2"));

        setCont(new Controller(this));

        setMg(new MainGUI(this));
        getMg().generateGUI();
        setBrickInControl(null);

        setFh(new FlockHandler(this, getBrickList()));

        restartServer();
    }
```

1

2

FIGURE 16: THE CONTENT OF COMPUTER.JAVA

The file will exactly look like Figure 16 and you will only need to modify:

- 1. The port on which you will communicate with the program in charge of controlling the drone and making the image analysis;
- 2. Add the bricks you will use. The GUI will dynamically create itself. You just really just need to specify all the brick you will work with by adding the line:

```
getBrickList().add(new Brickb(brickID, this, brickName, brickMACAddress));
```

6.2.4.2.1 LAST (PRO-)TIP BEFORE LEAVING

Forgetting what library to include is a common mistake so here is how your Java build path should look like on Figure 17 (if you are using Eclipse, if not, you should be able to extract the information you need).

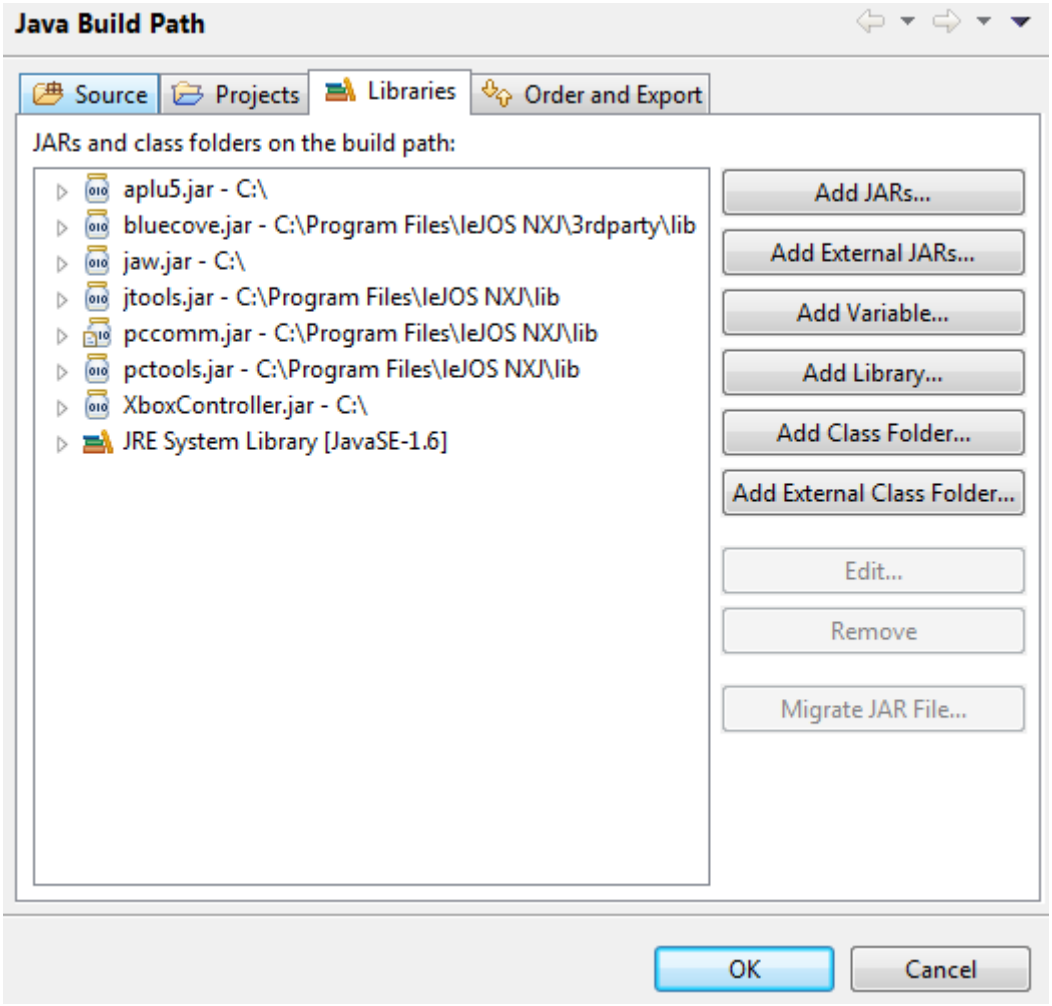


FIGURE 17: JAVA BUILD PATH FOR THE COMPUTER CODE

6.2.4.3 ANY MORE QUESTIONS?

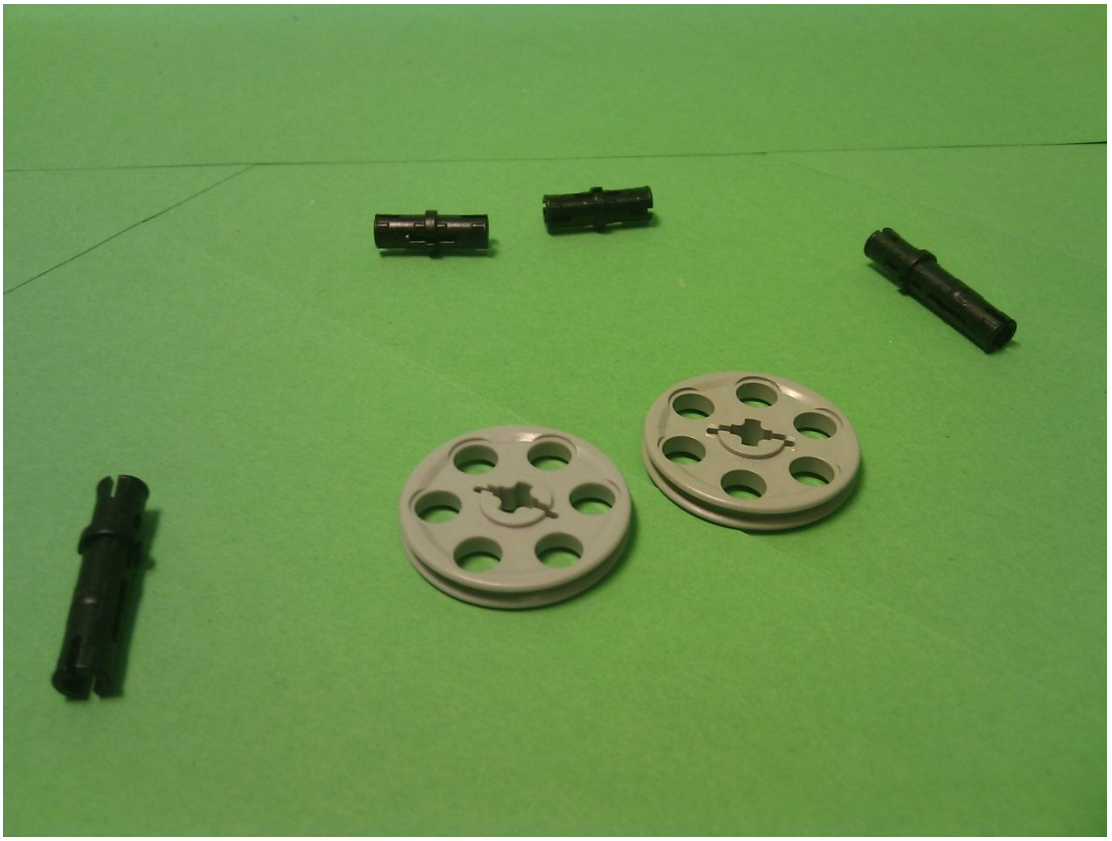
Check our website for more information; otherwise feel free to e-mail us: guillaume@ludep.com

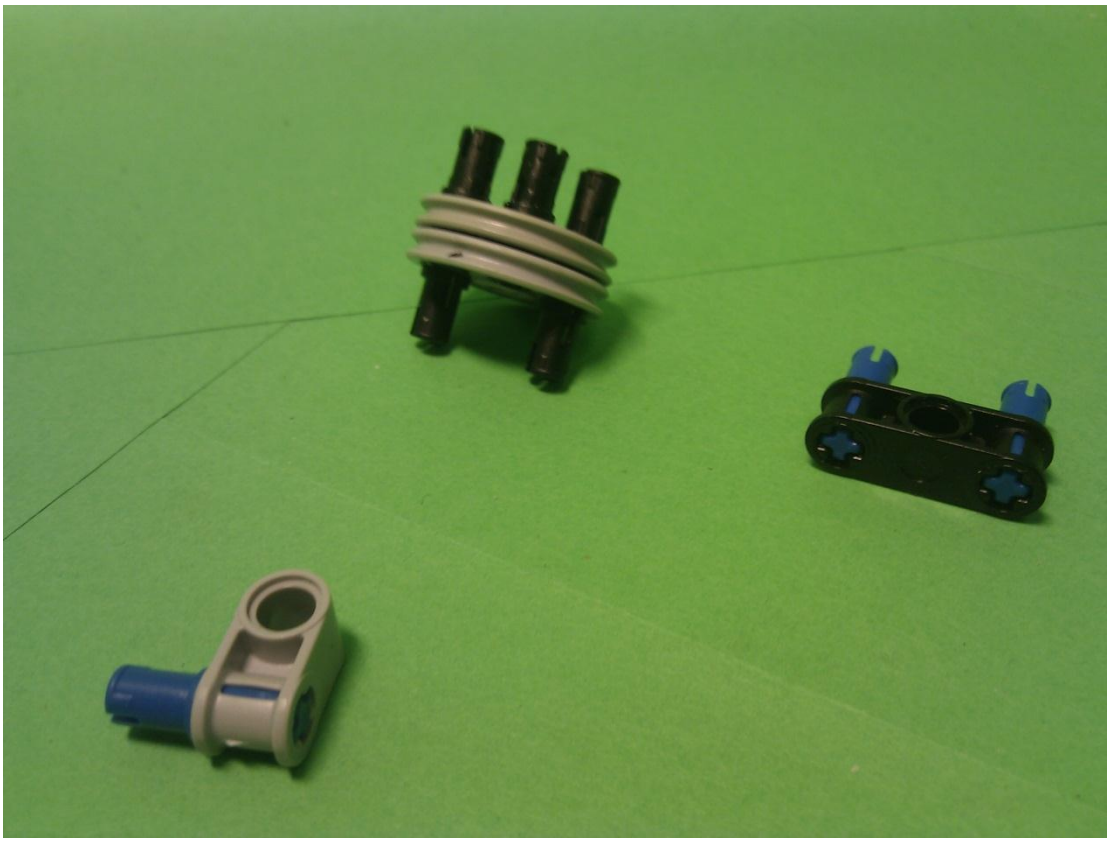
6.3 APPENDIX C: BUILDING INSTRUCTIONS FOR THE OMNIWHEEL ROBOT

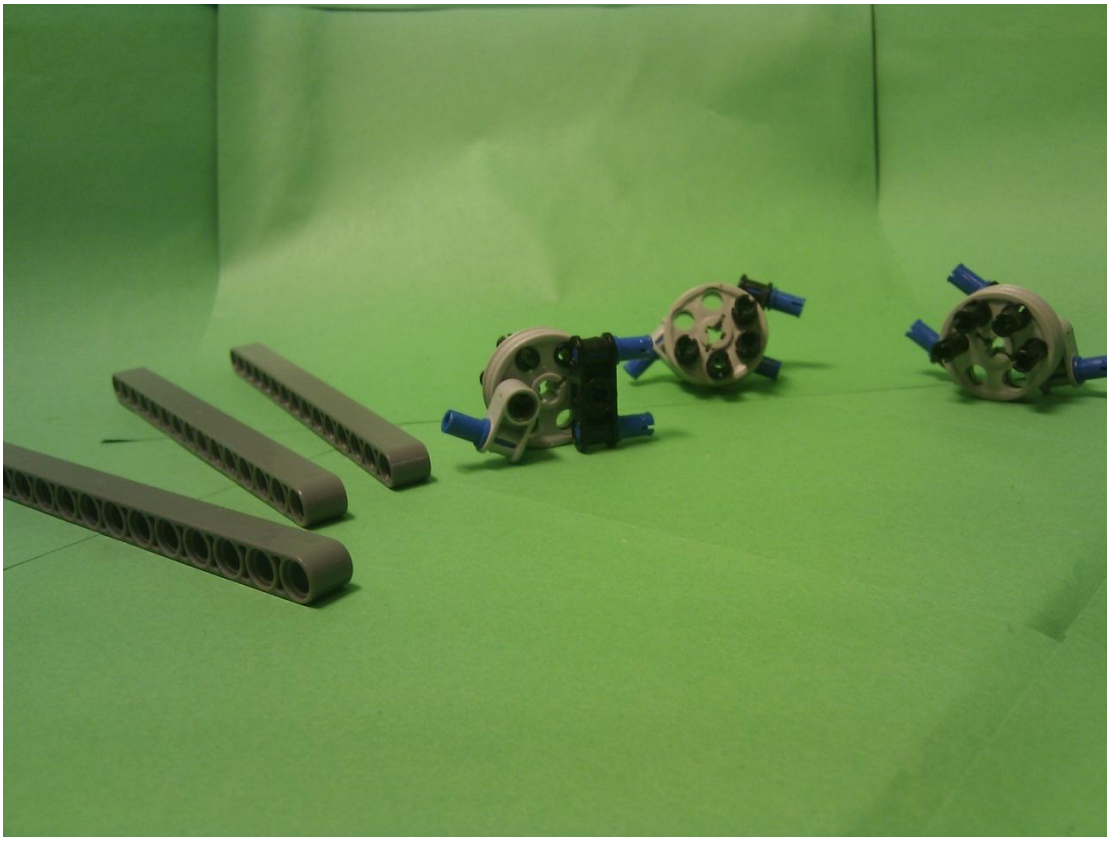
As we spent a really long time on the construction, the design of our robots, we thought relevant to deliver some instructions if anyone wanted to build them on their own. It should be detailed enough, good luck if you are attempting to do it.

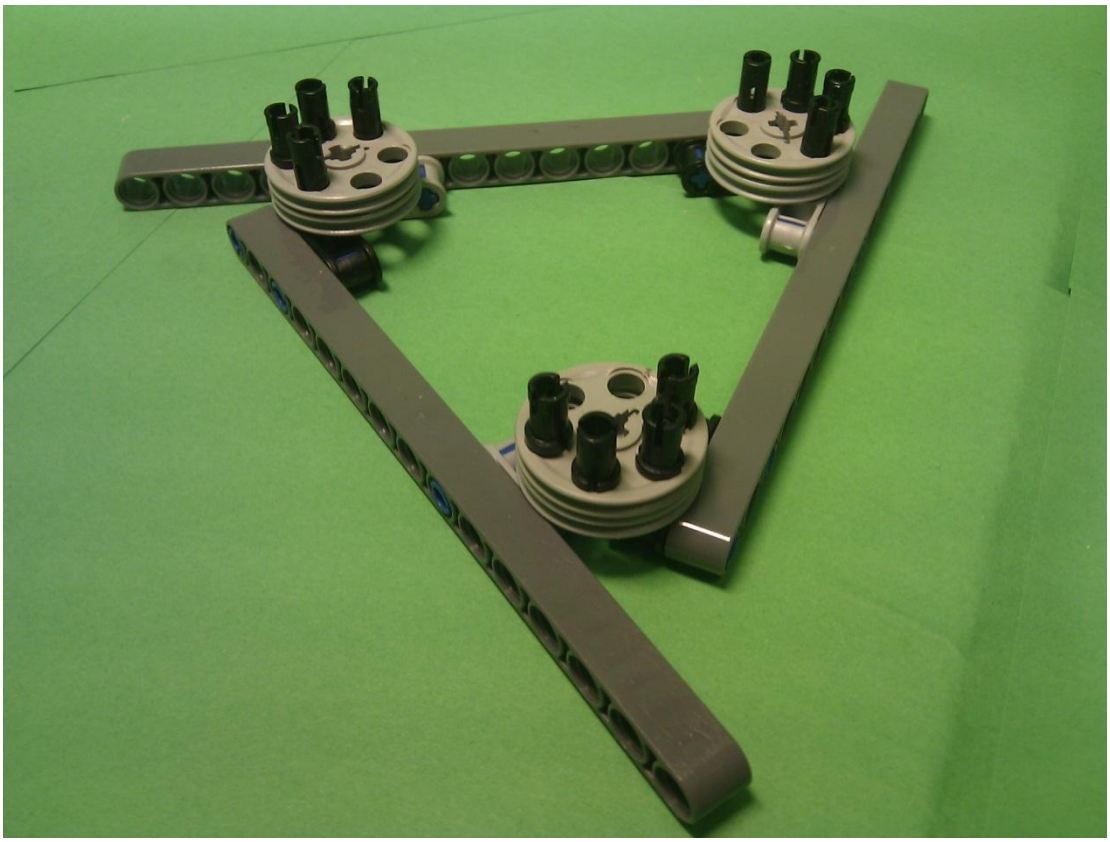
Please note that the only none-LEGO elements you will require are the Rotacaster omniwheels.

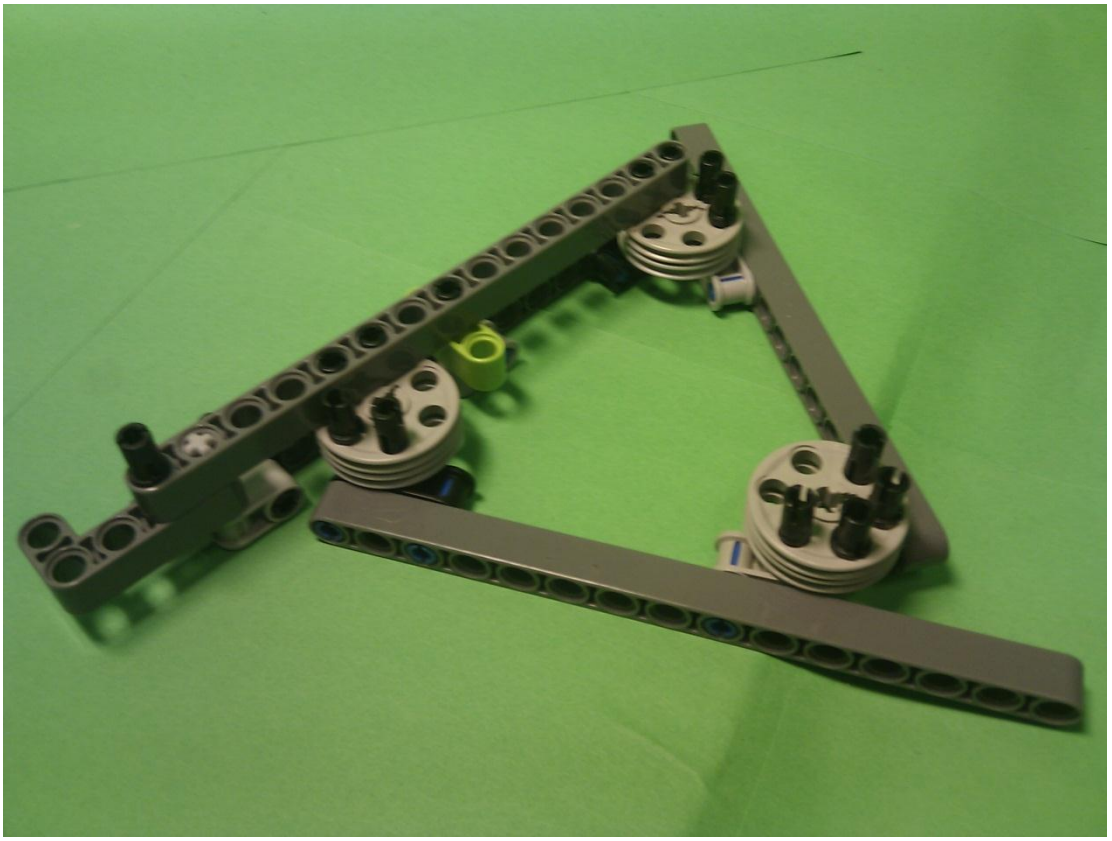
This is where it all begins…

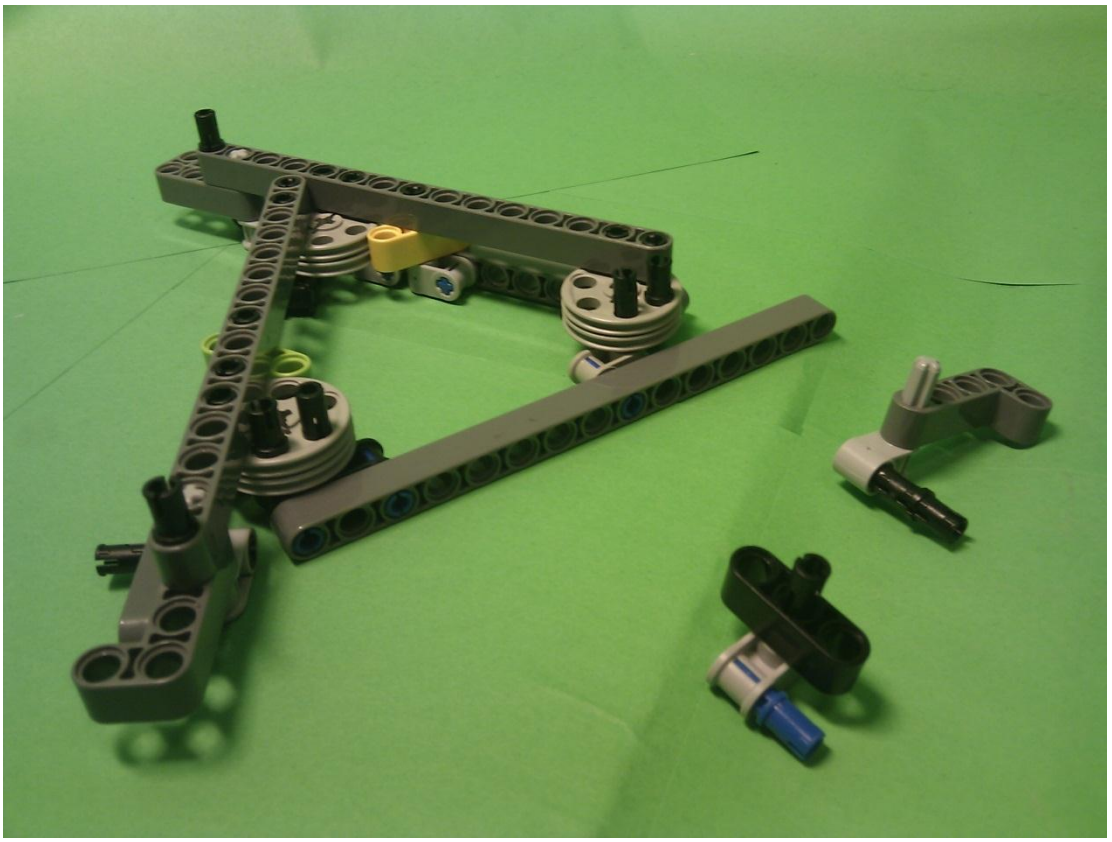


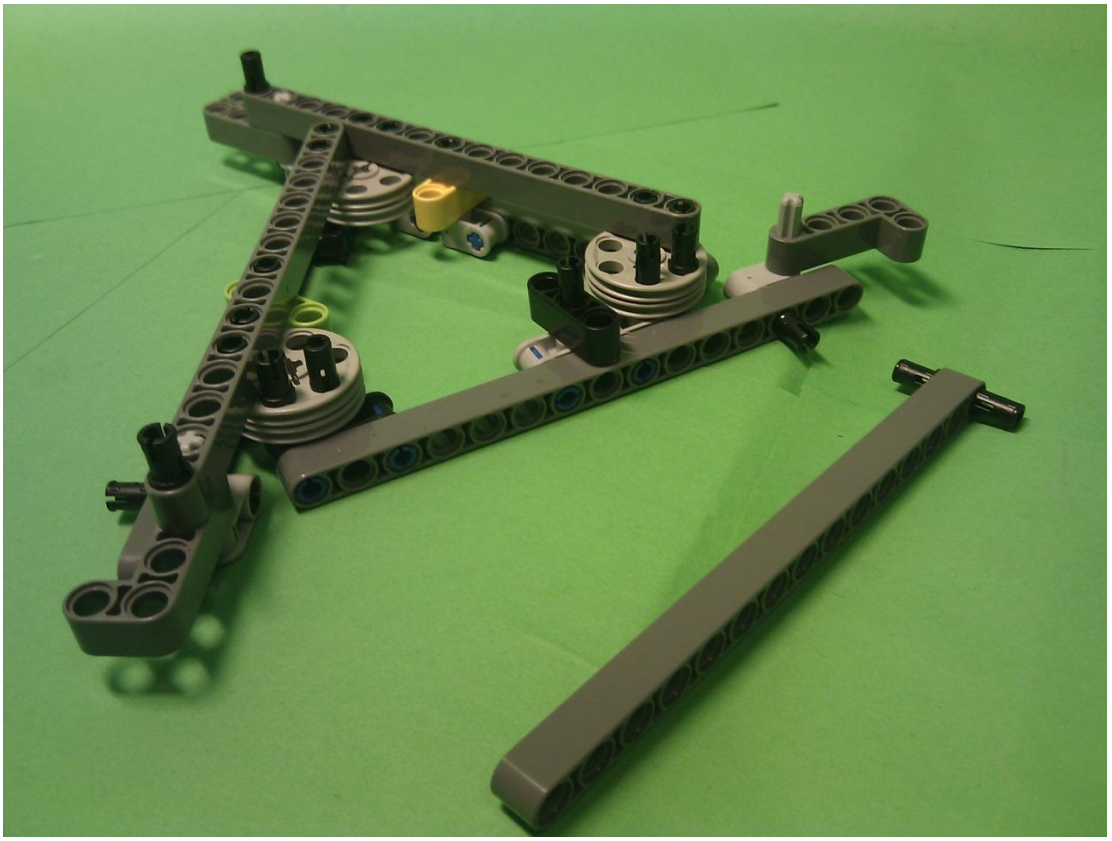


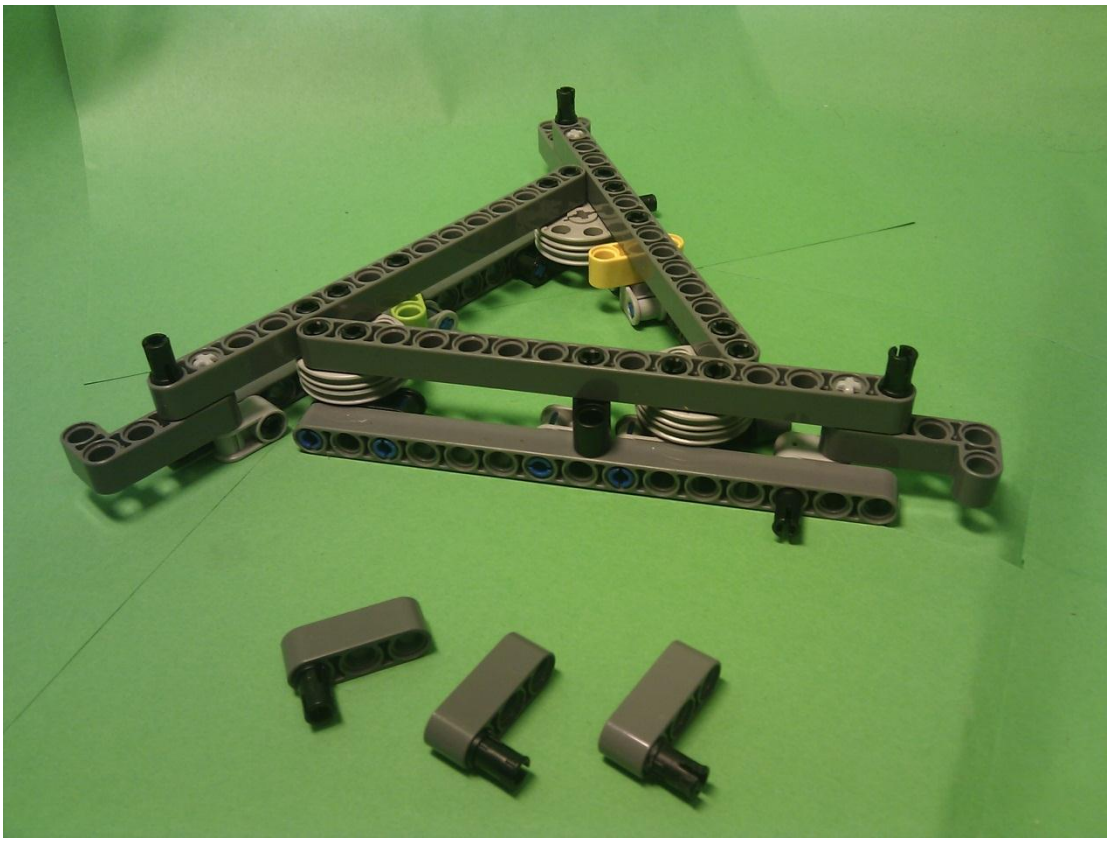


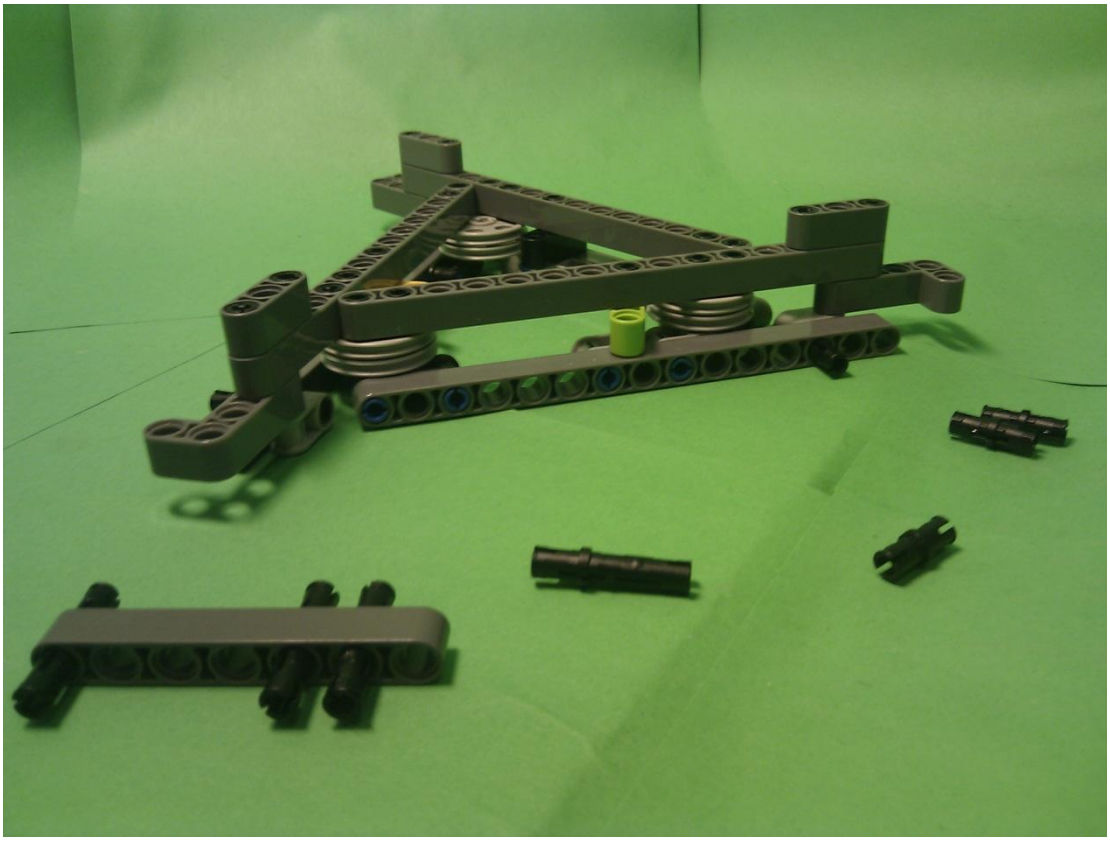


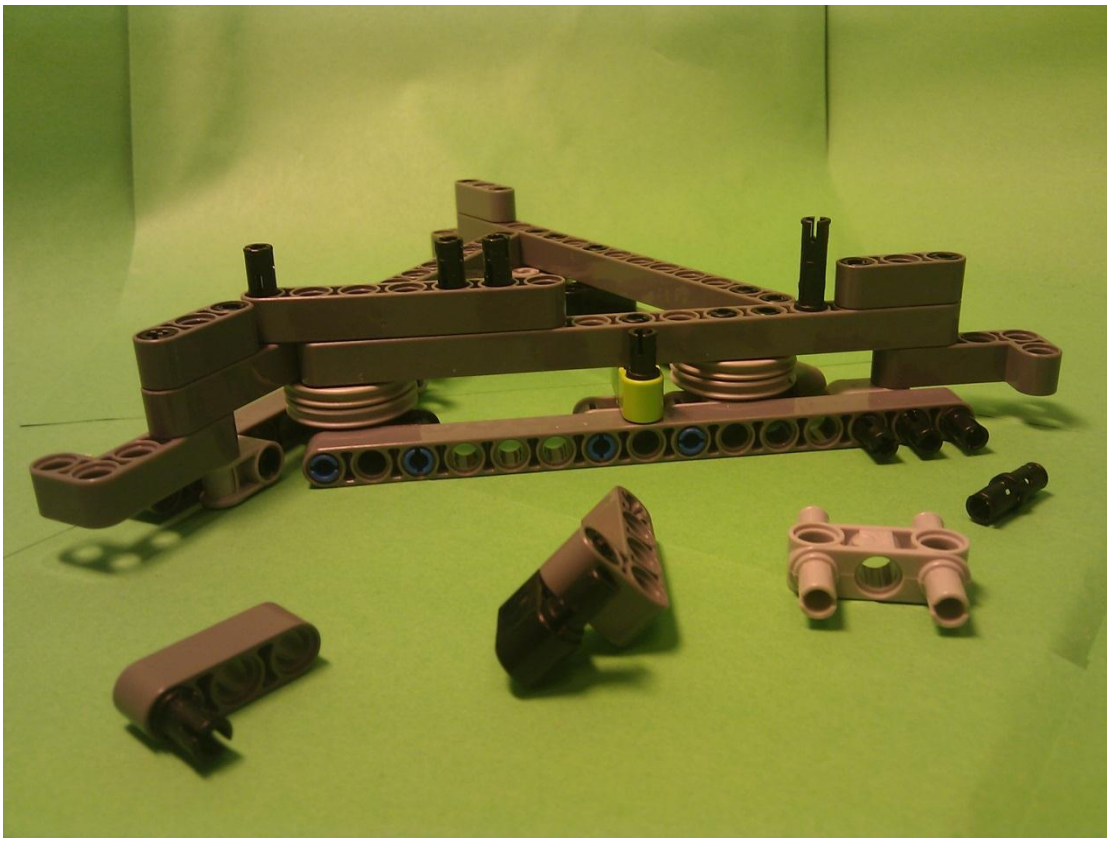


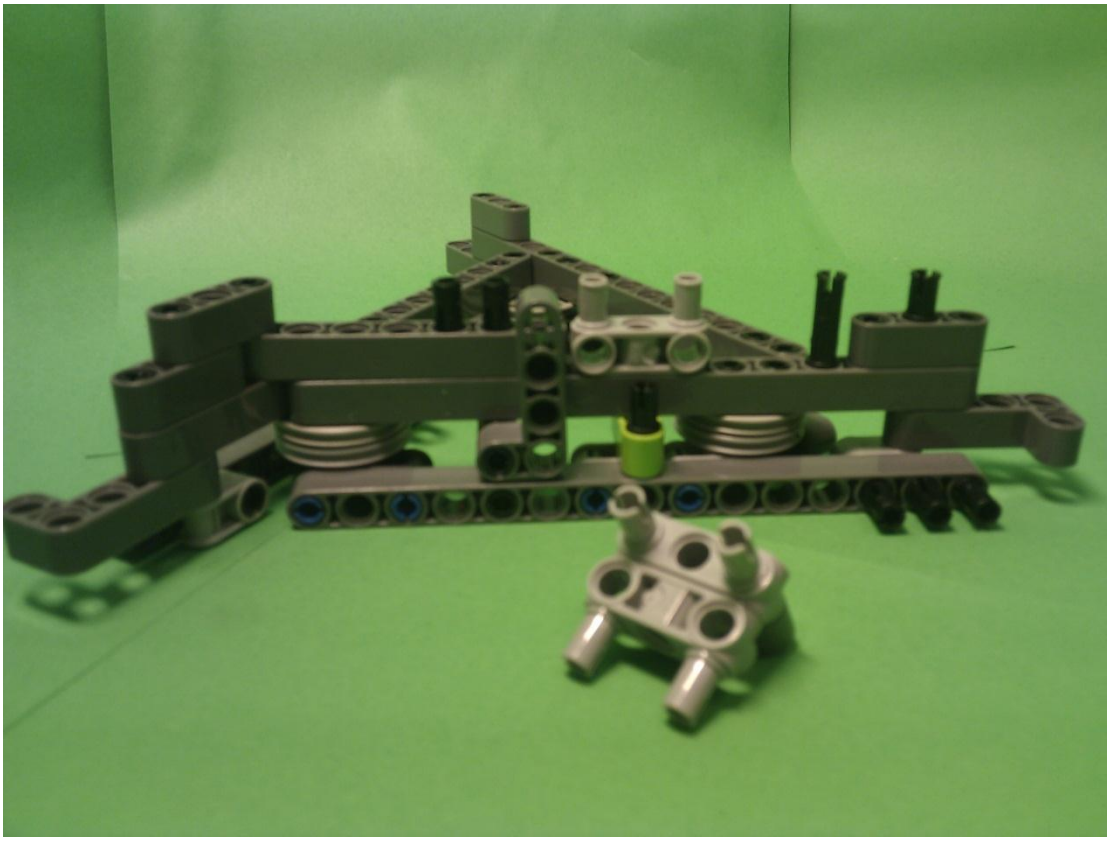


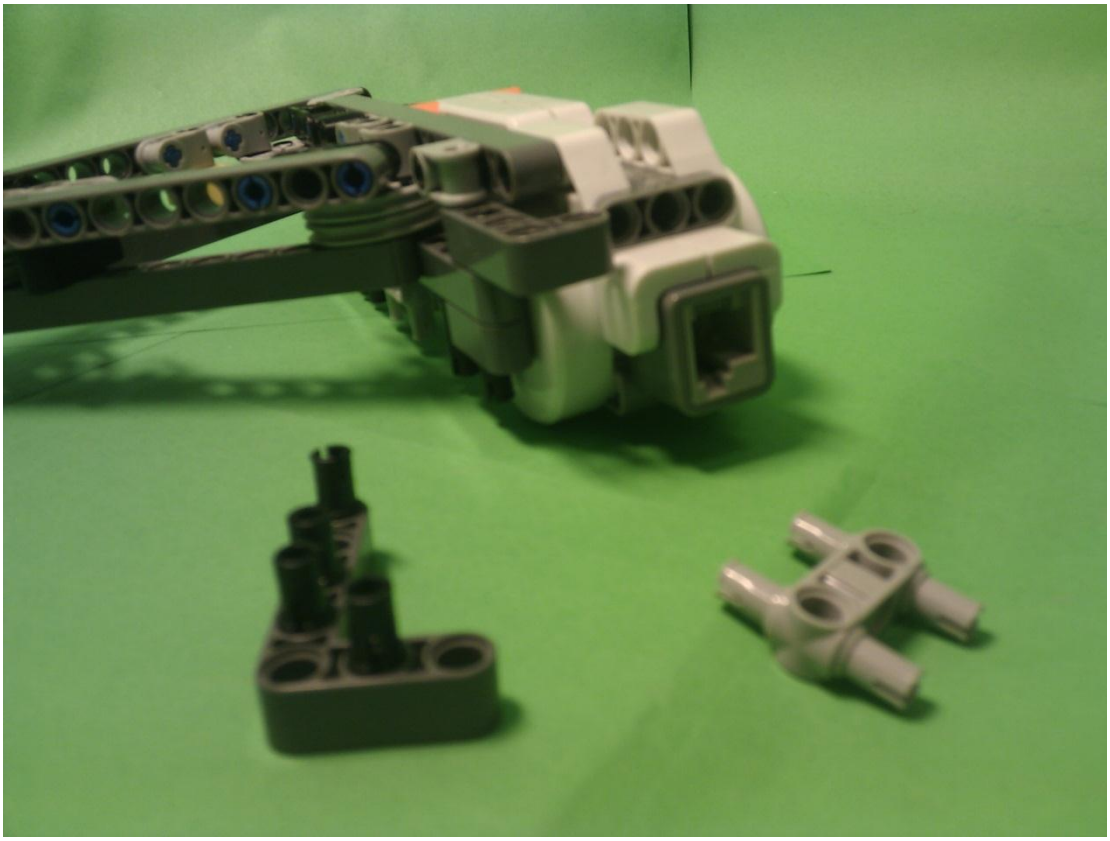


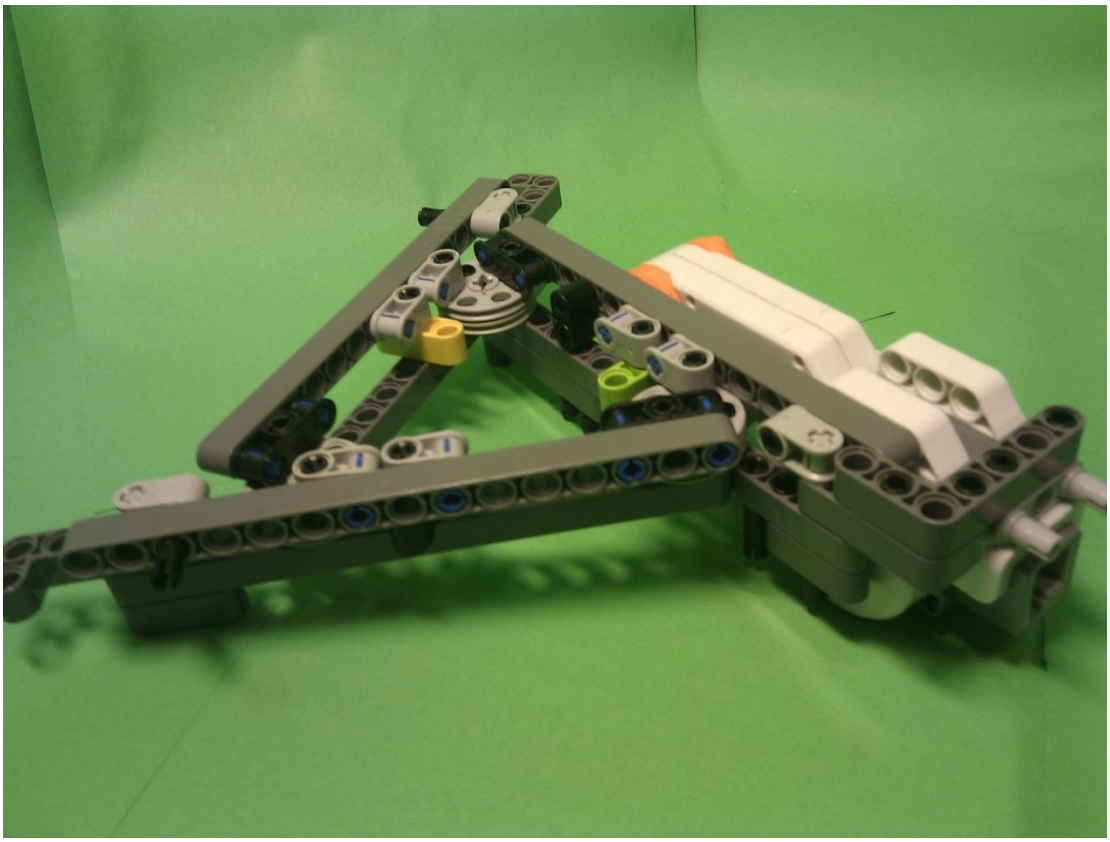


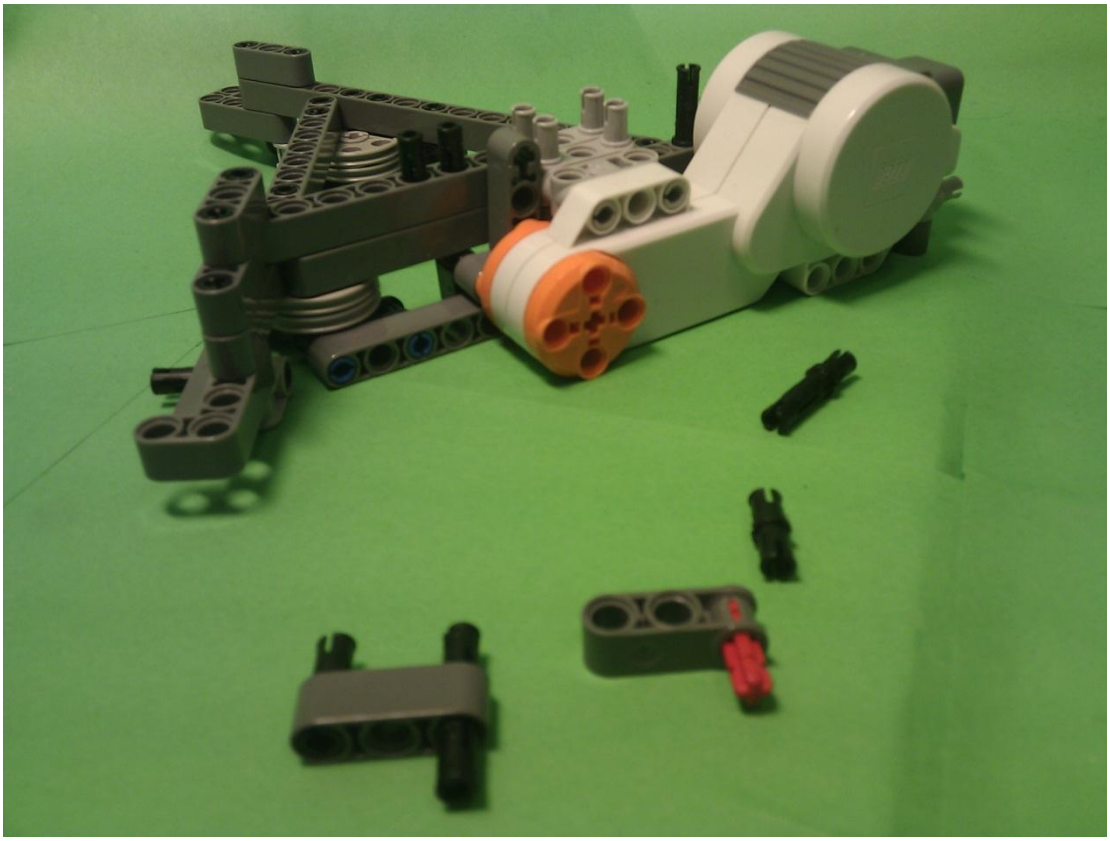


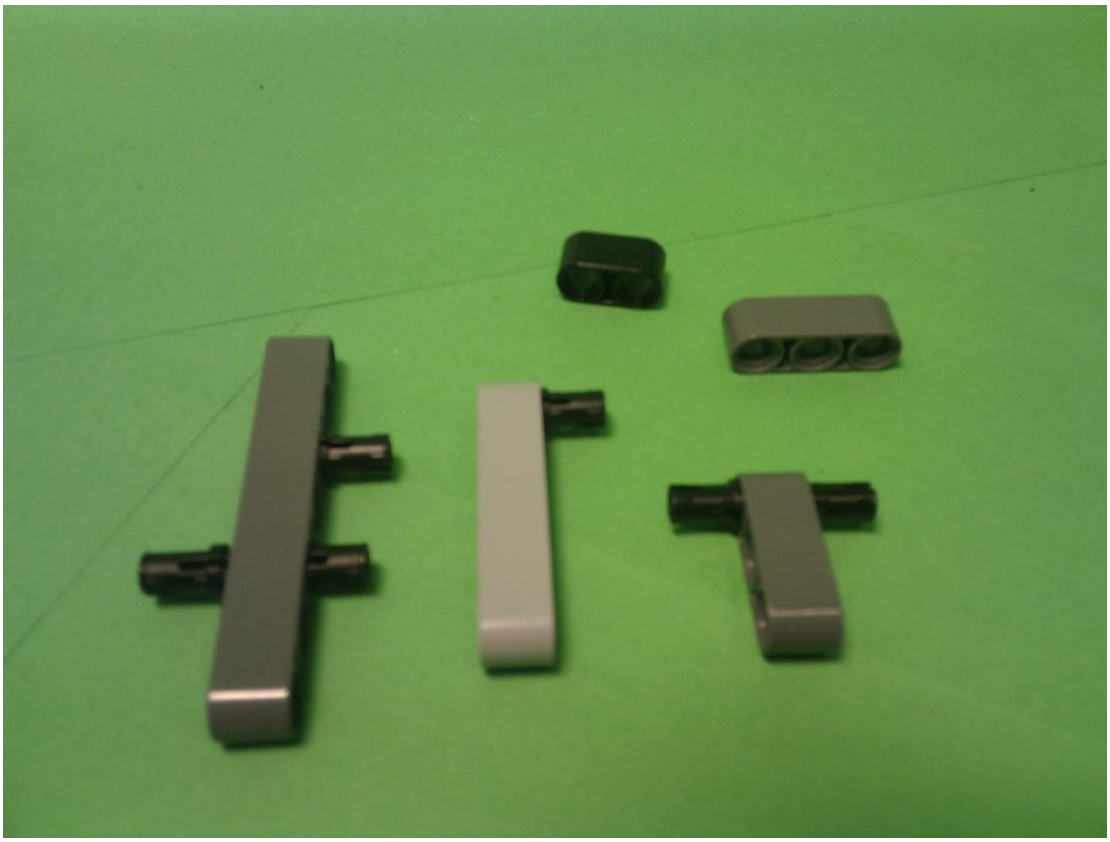


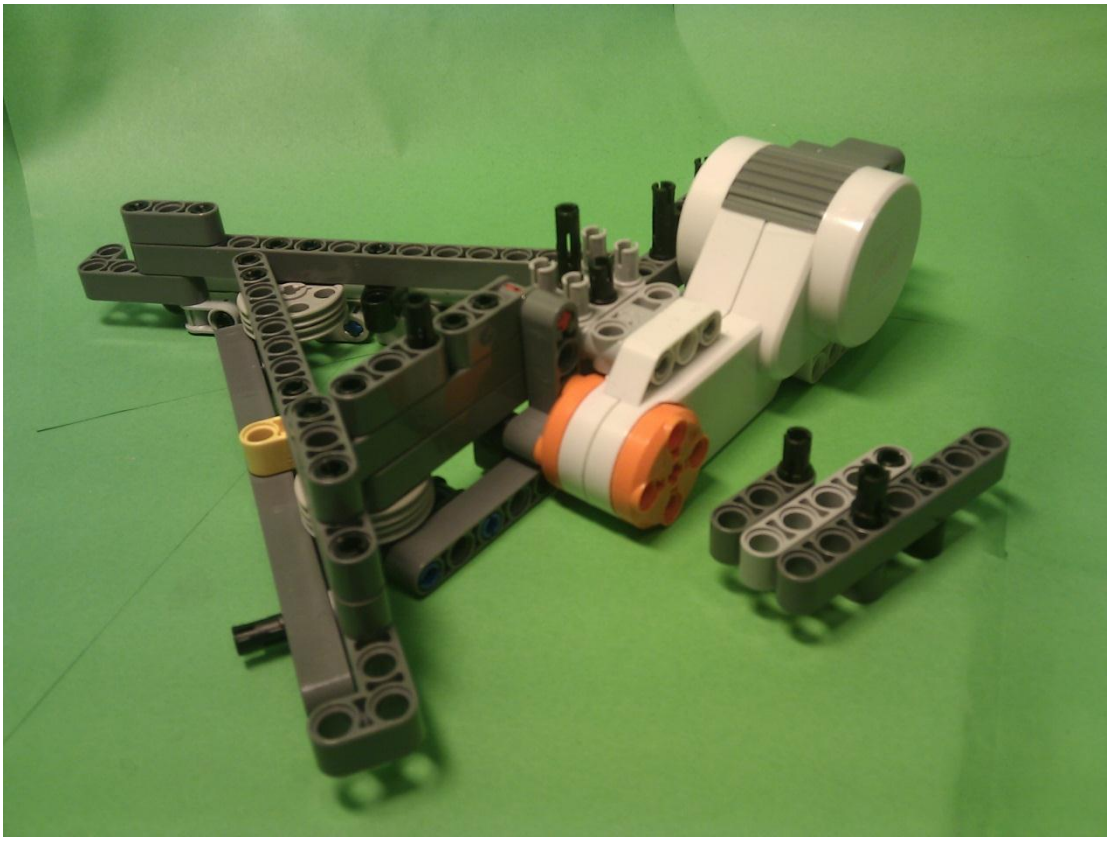


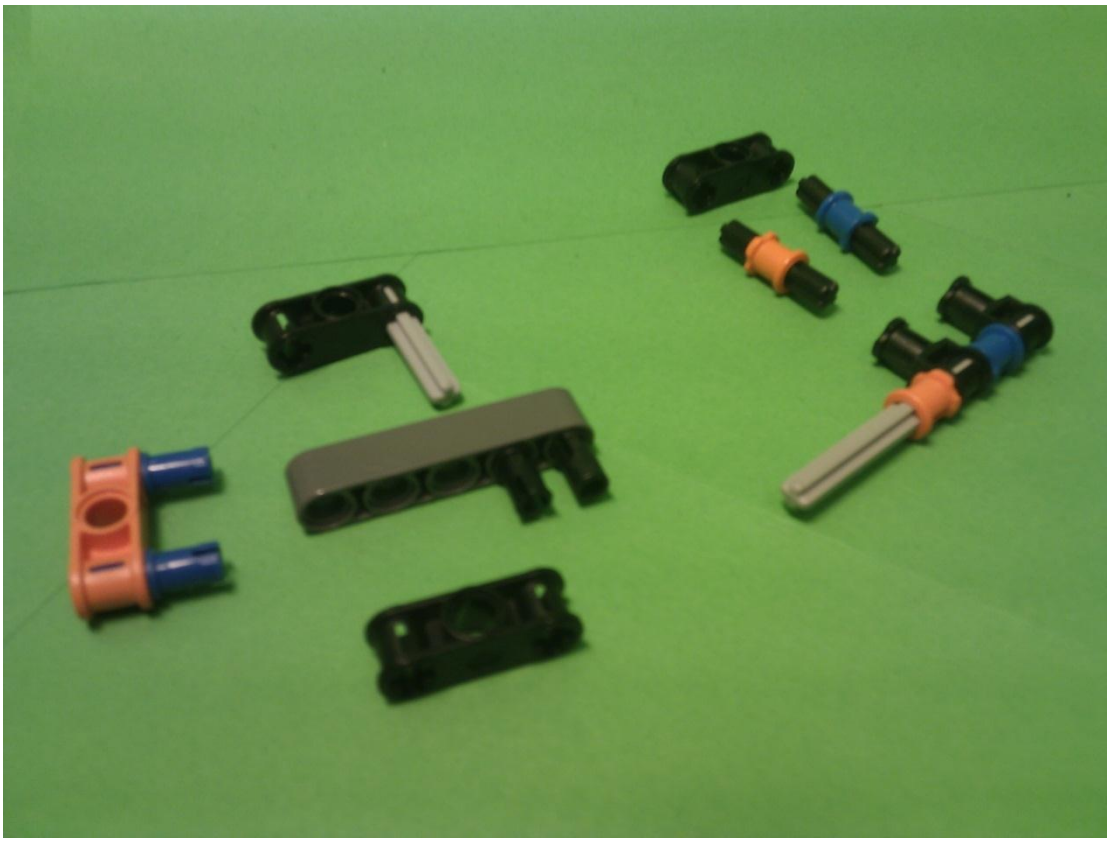


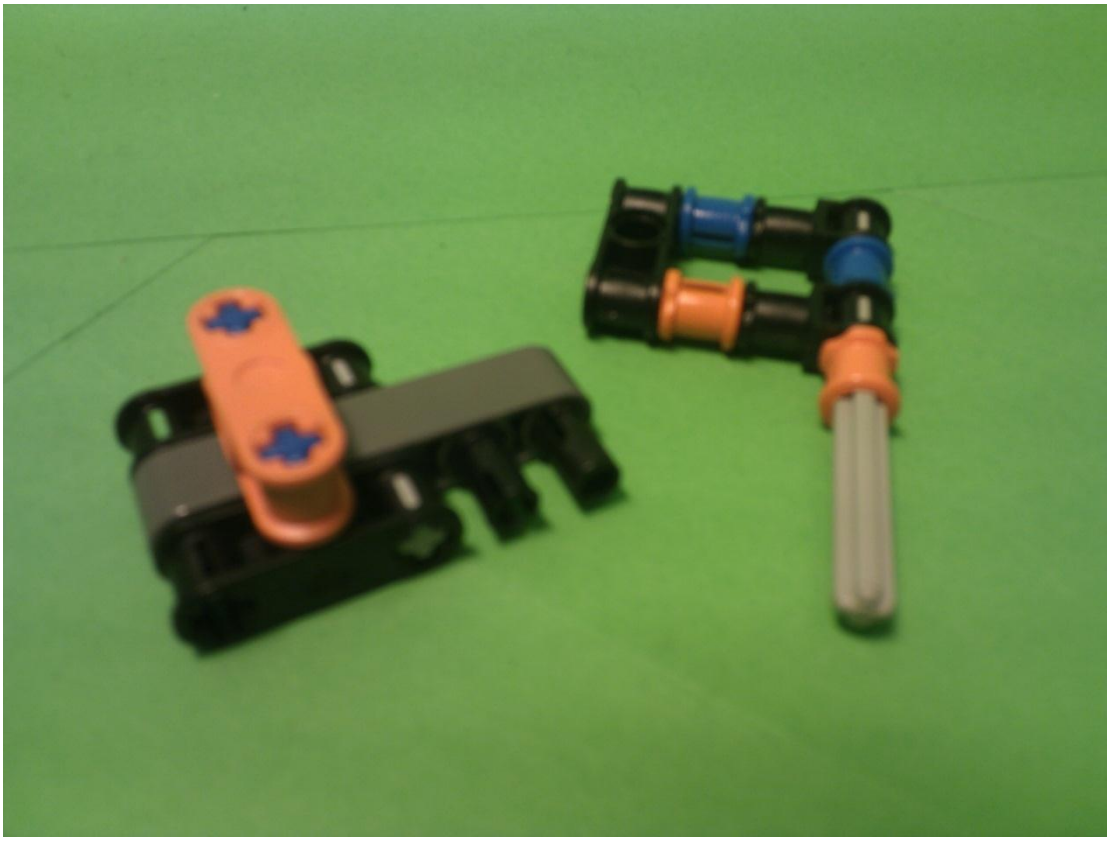


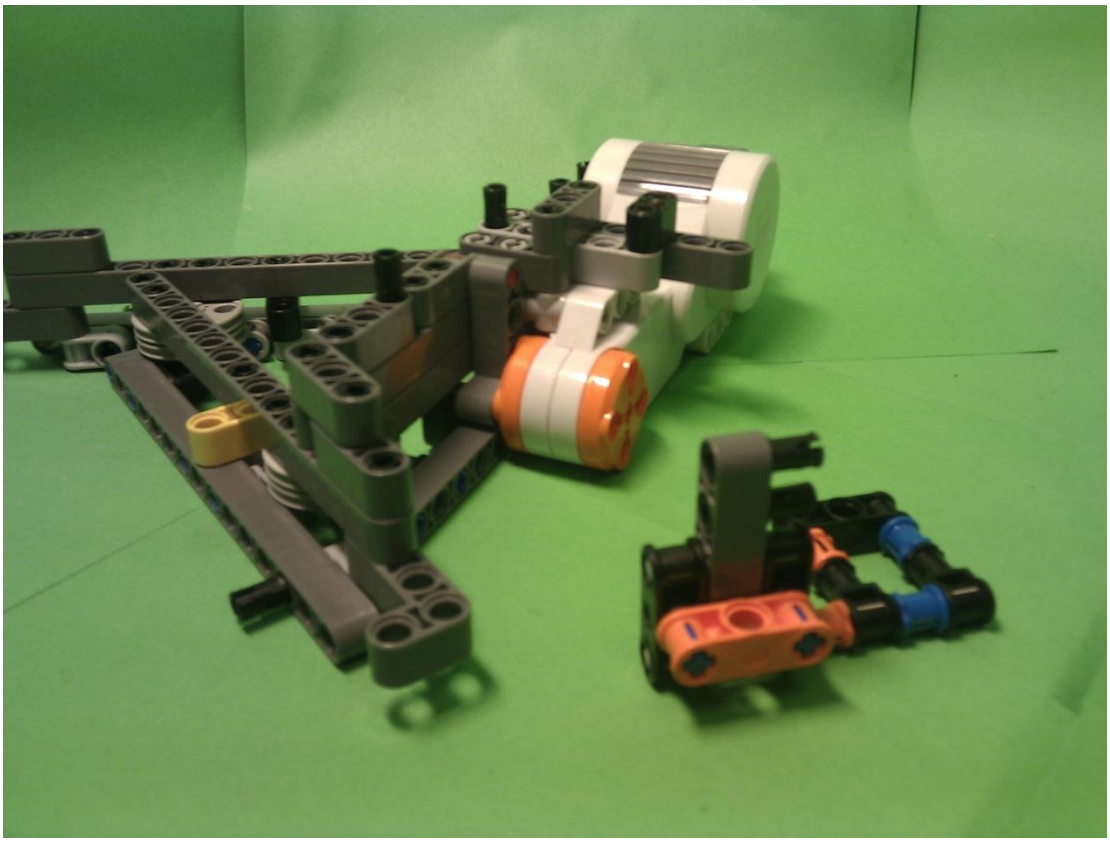


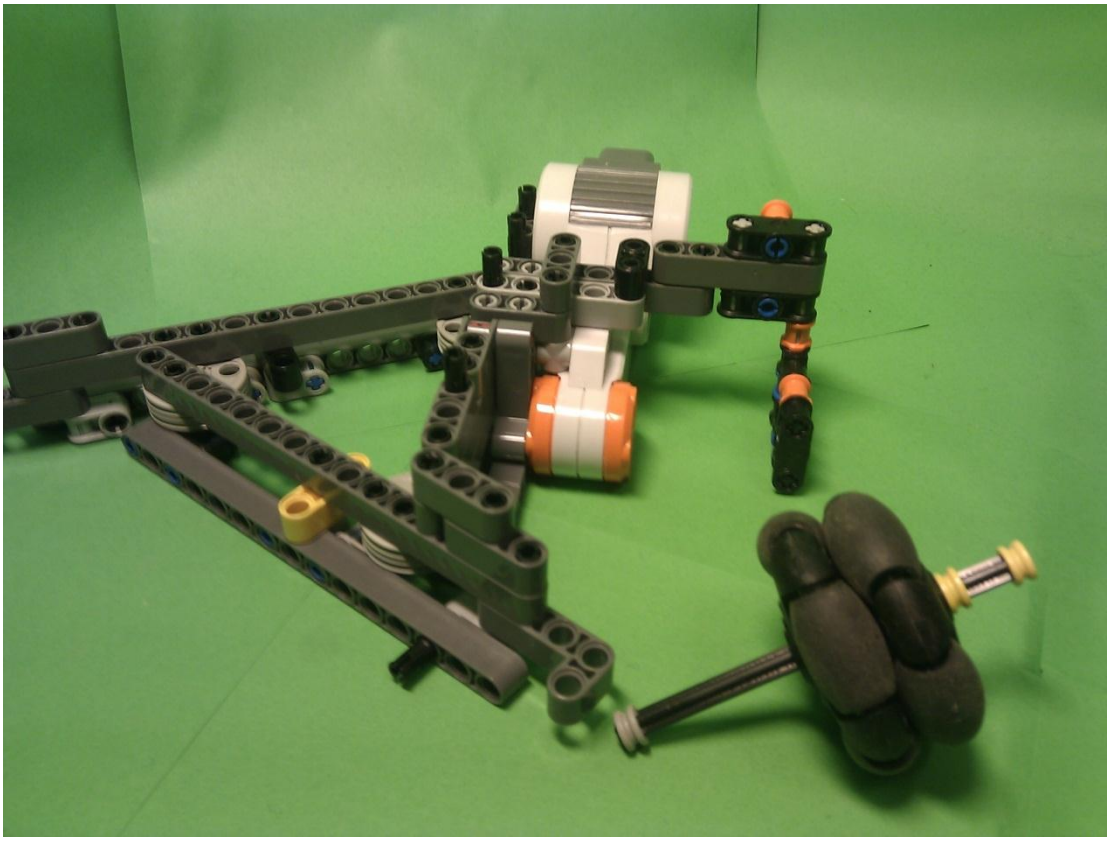


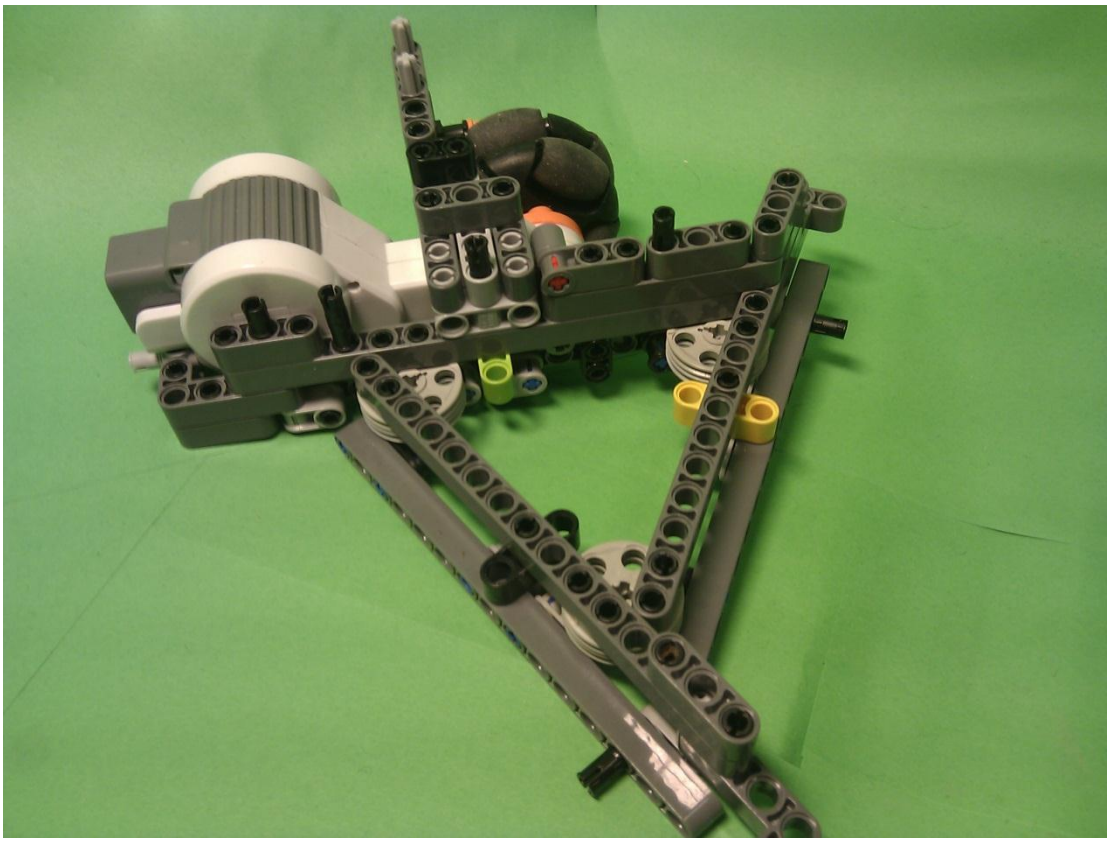


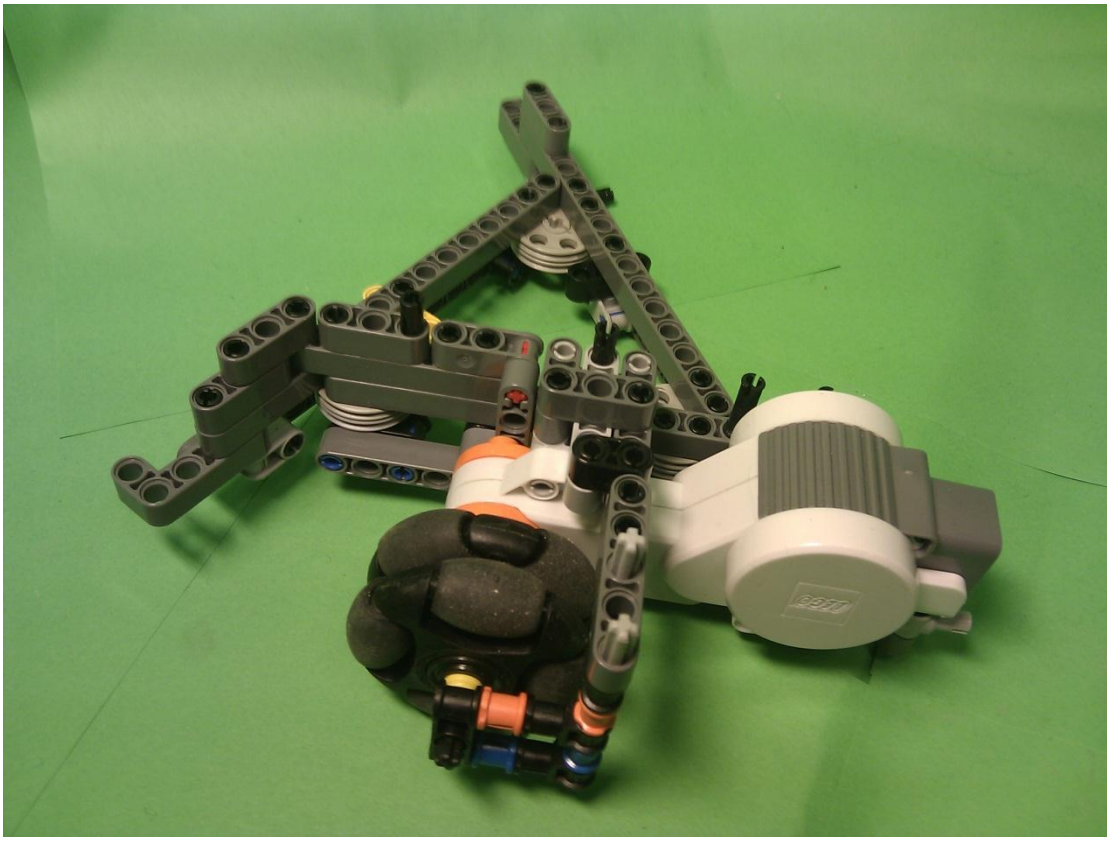


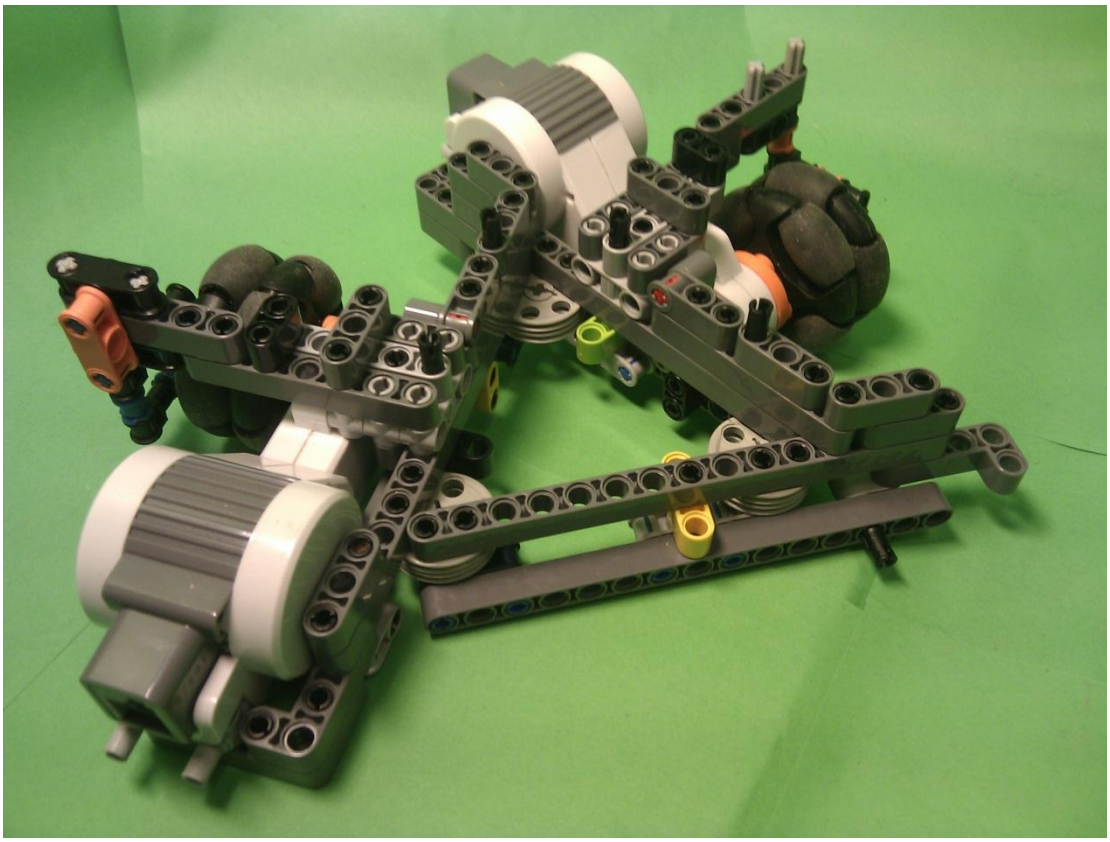


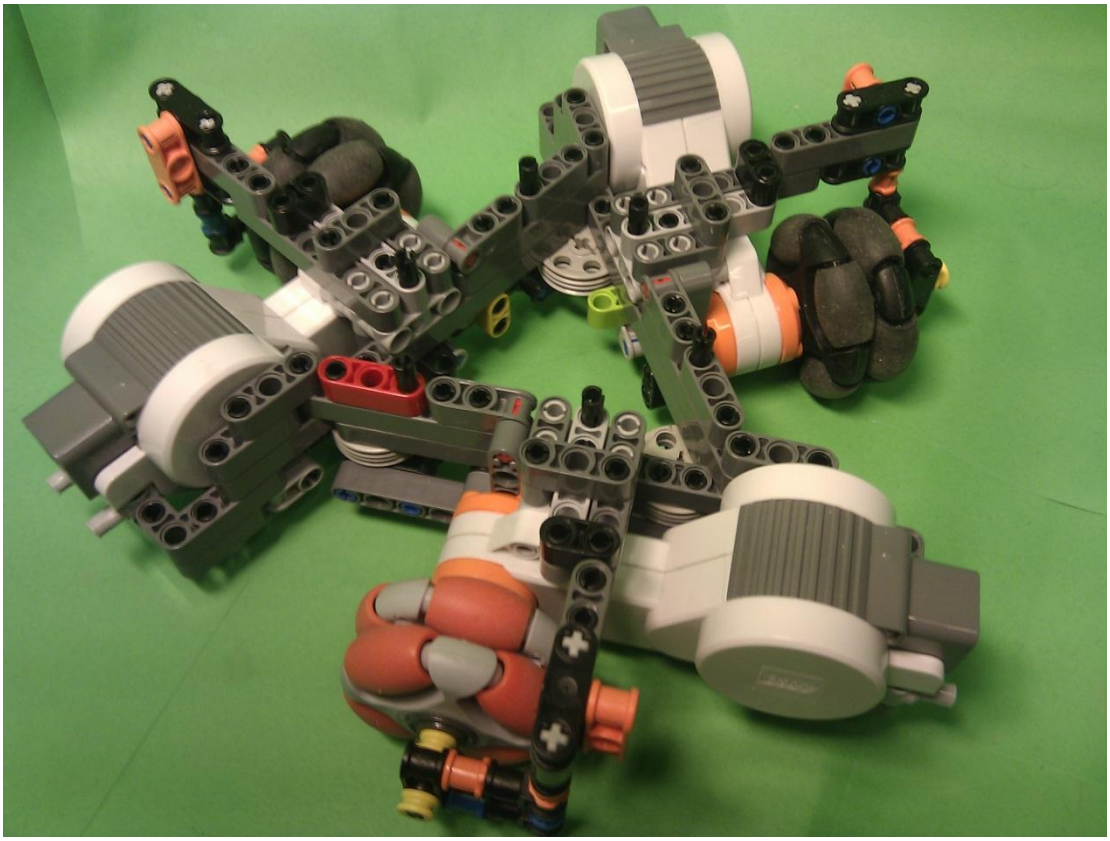


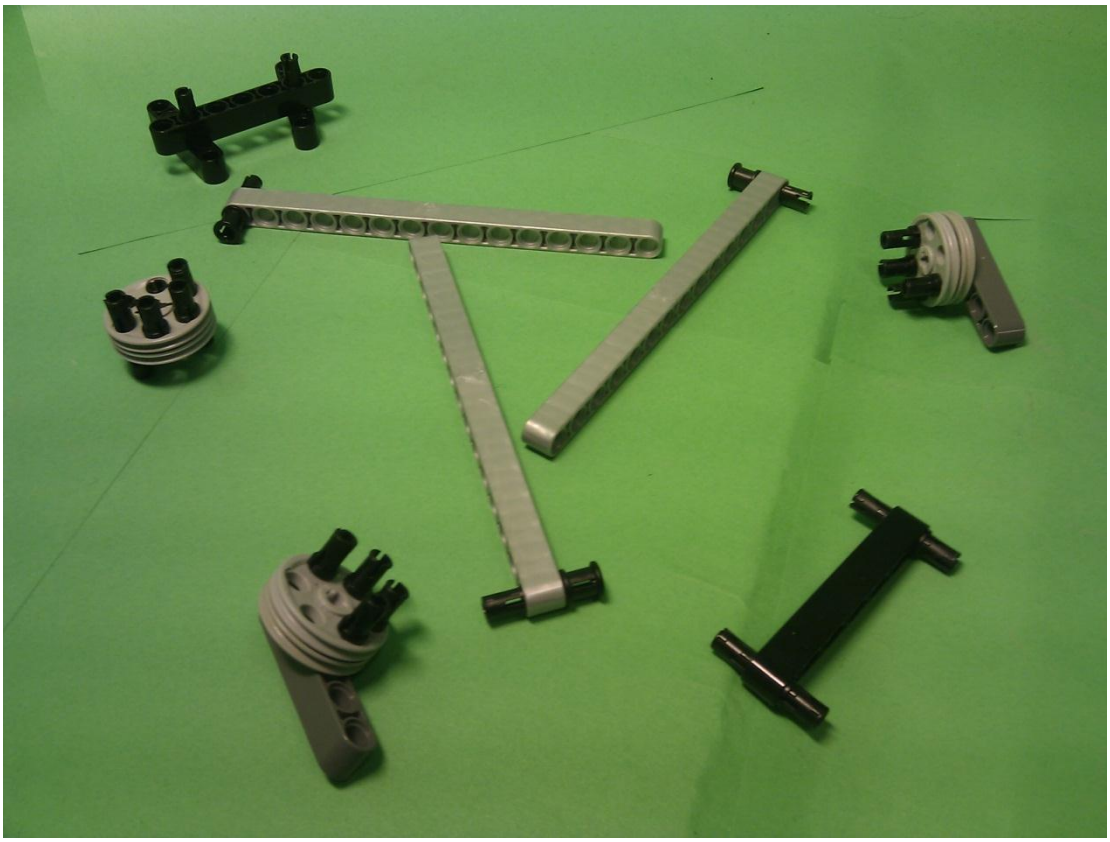


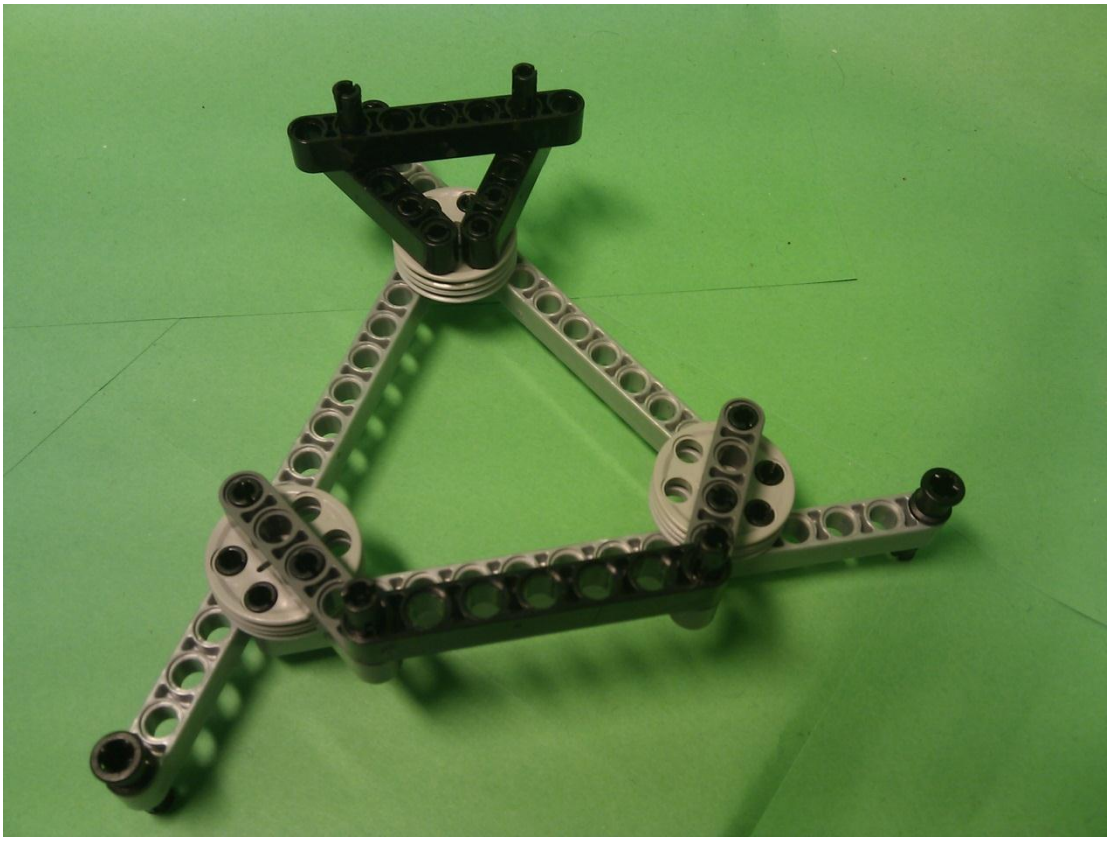


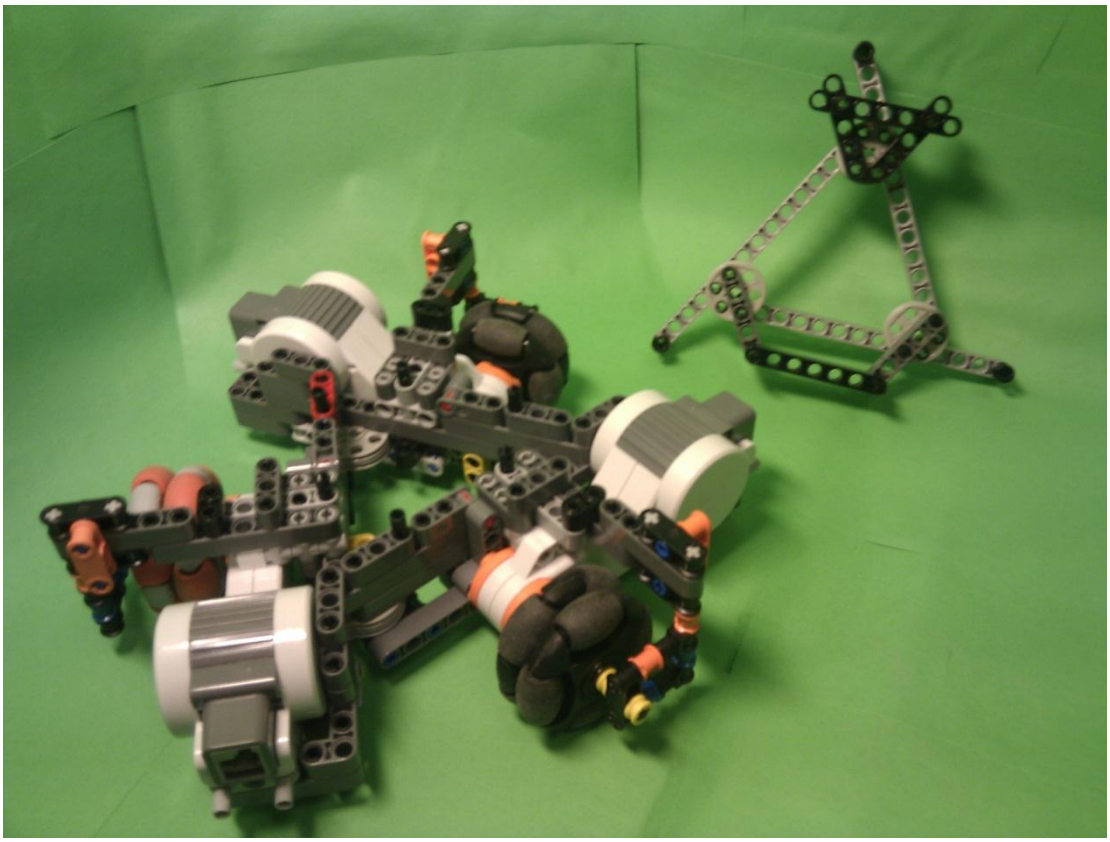


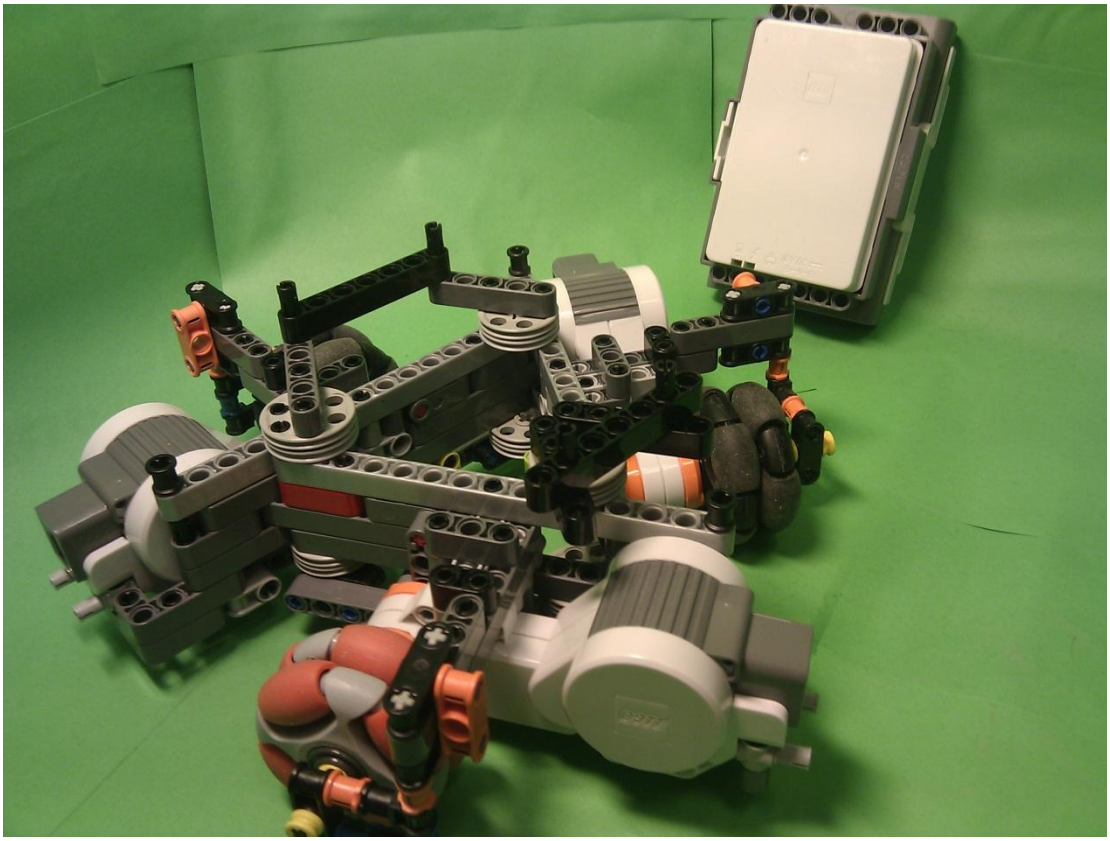


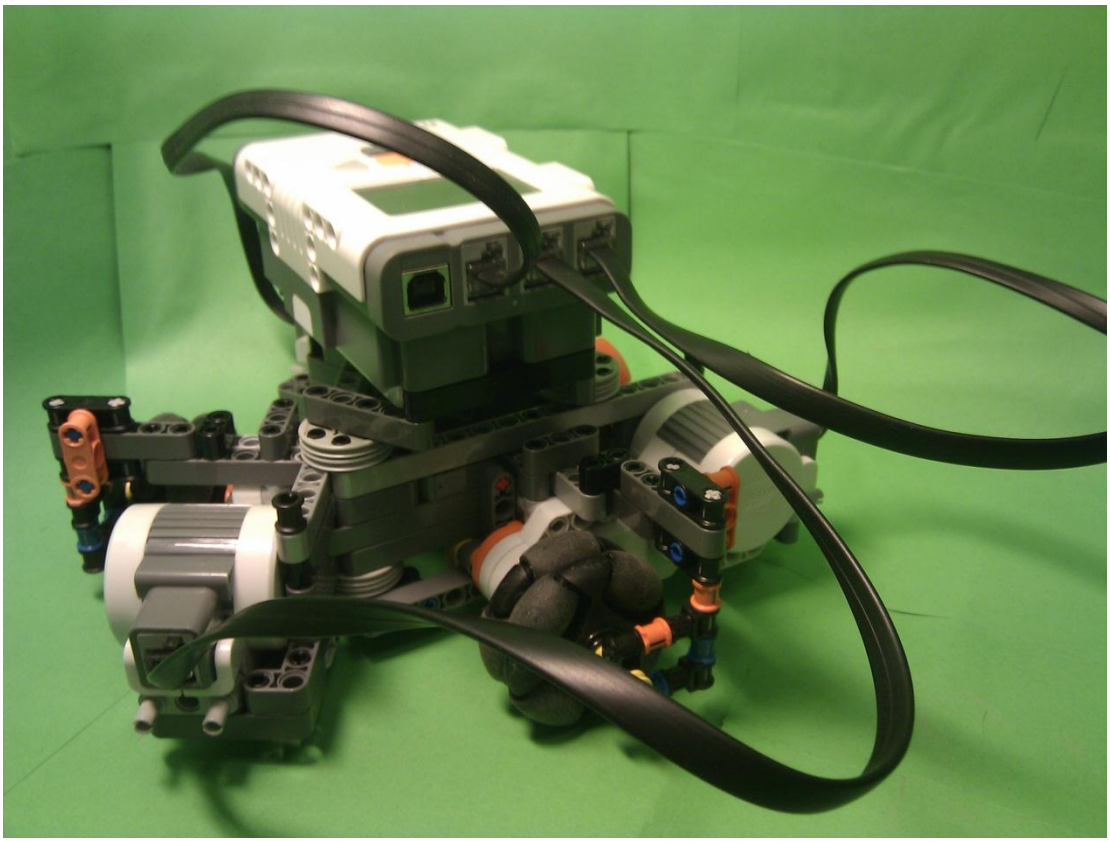


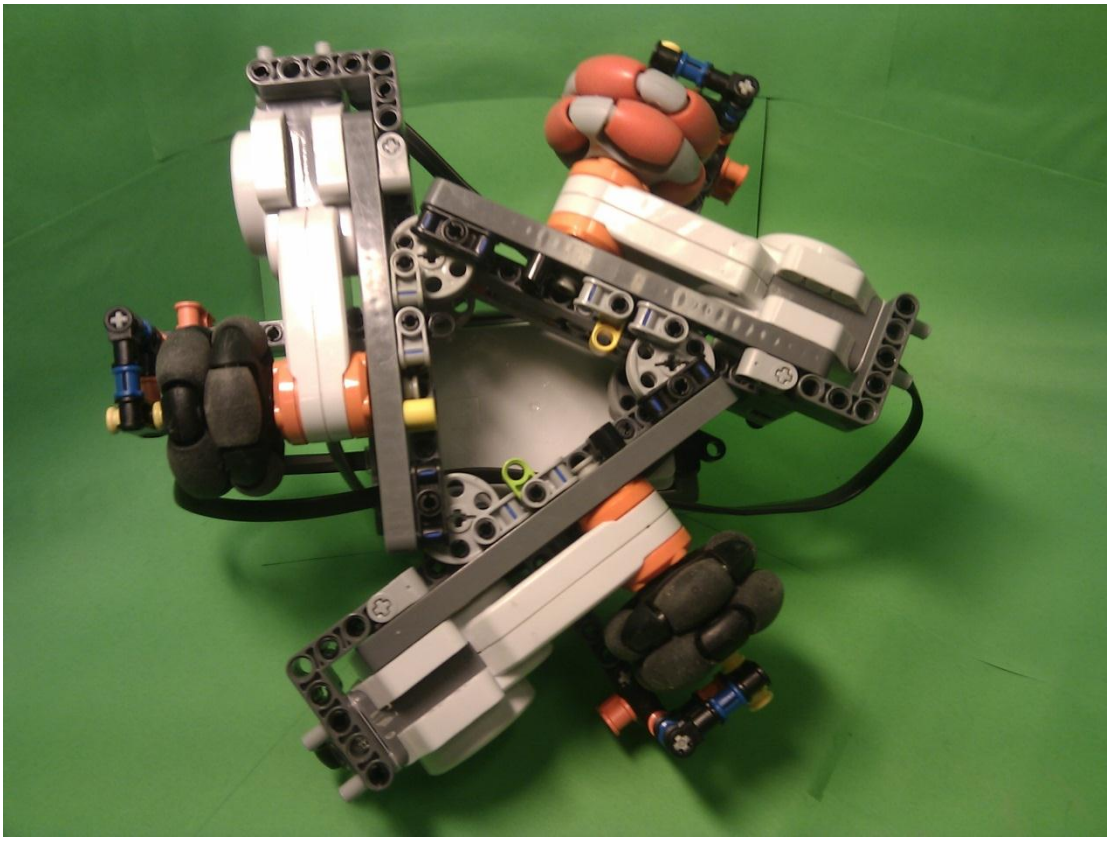


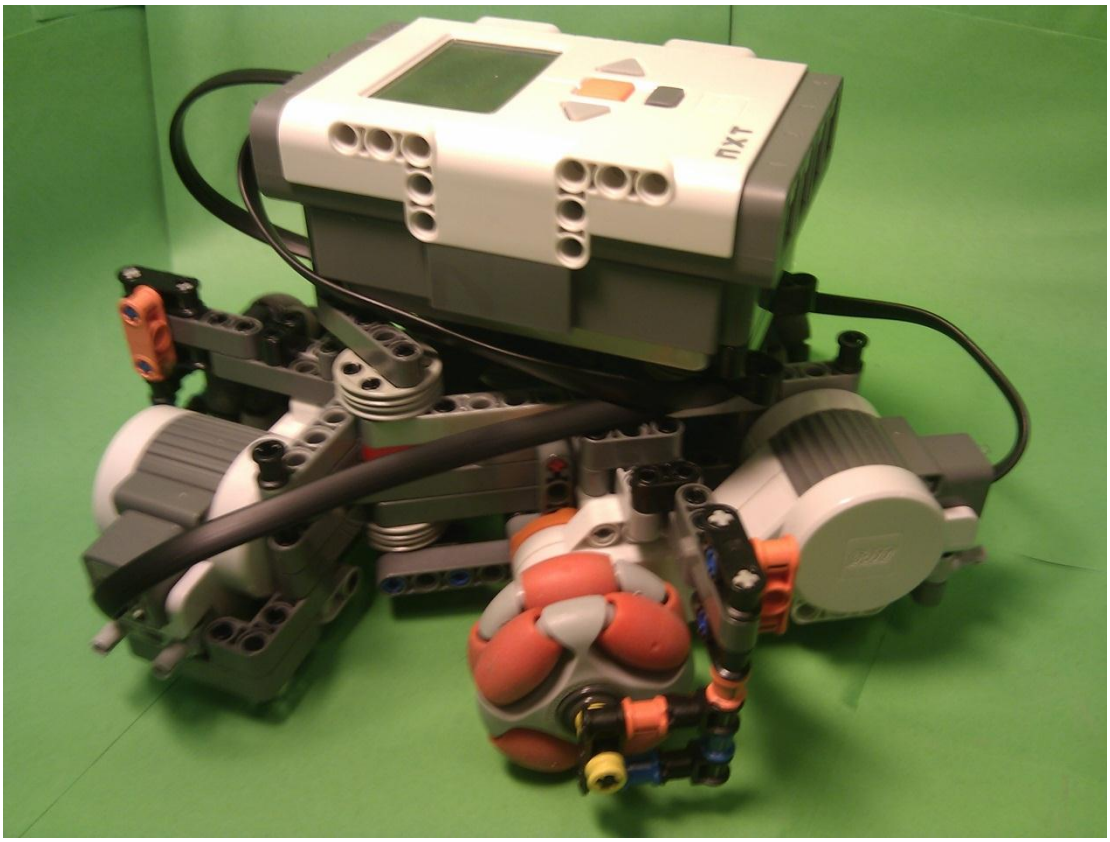


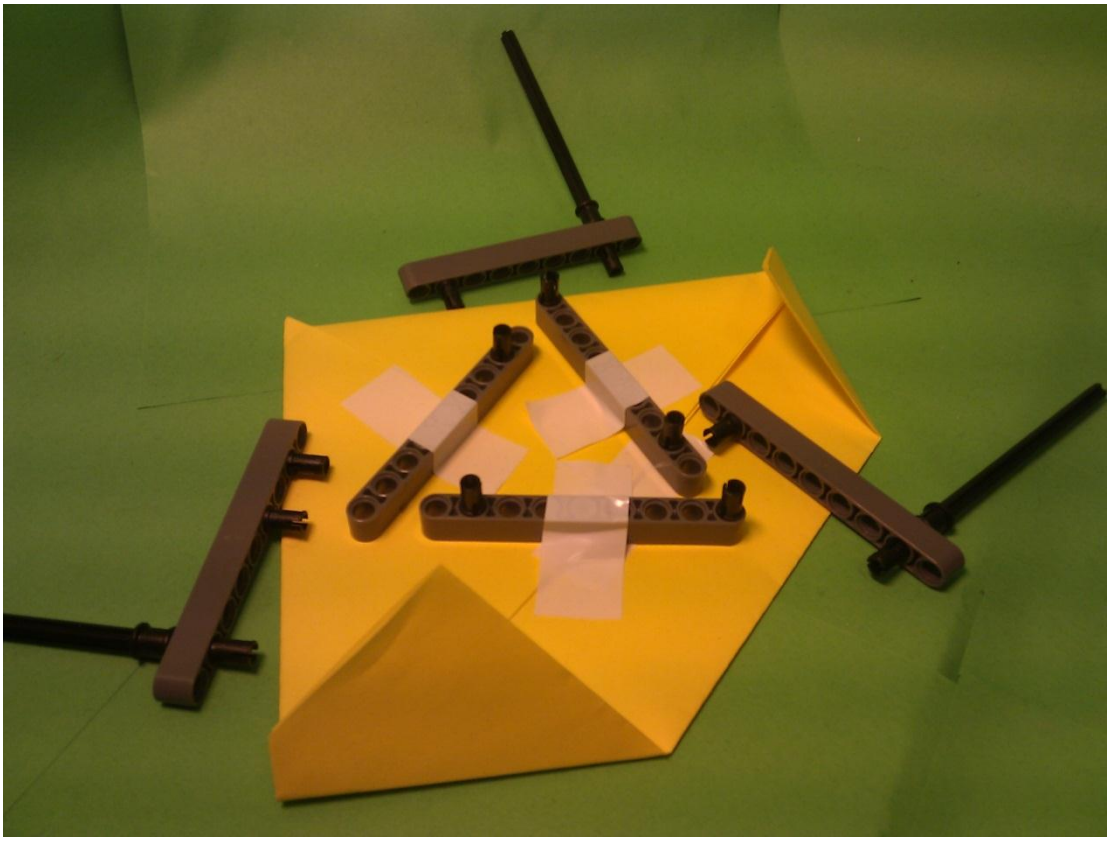


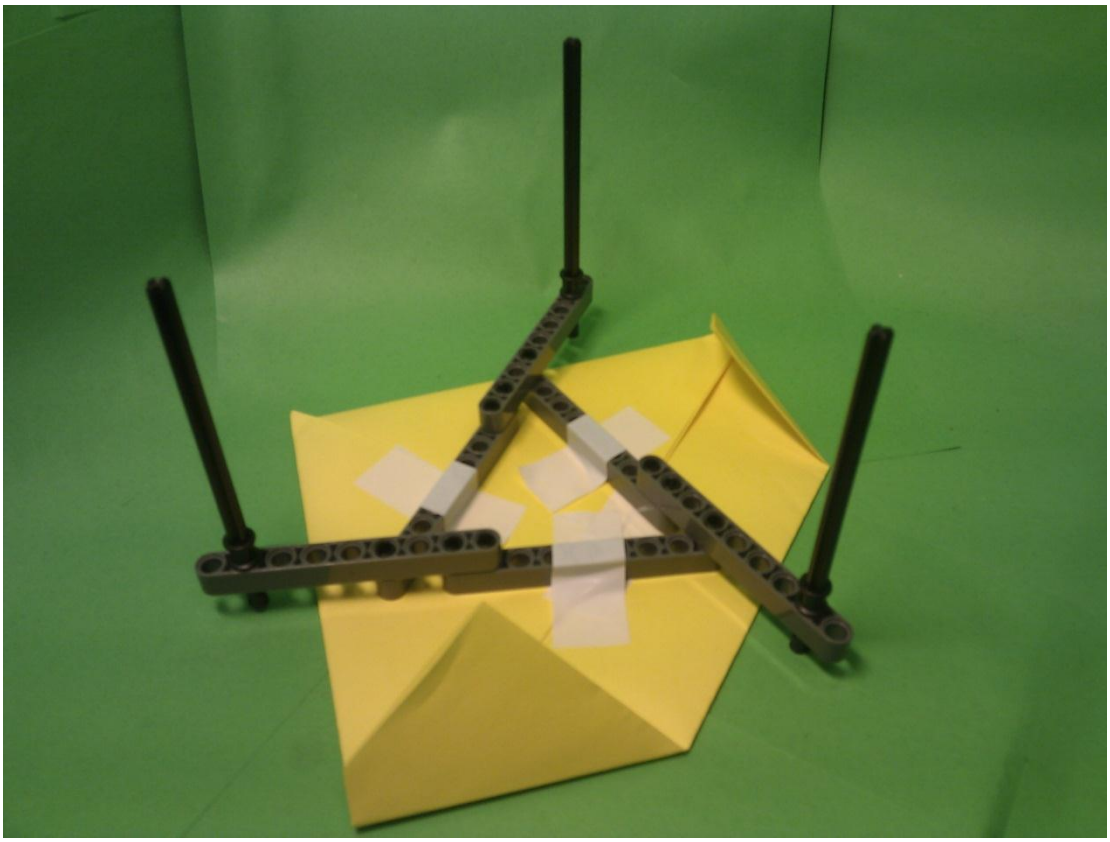







NXT





And then put this on the robot for image analysis purposes

7 ACKNOWLEDGEMENTS

In this section, we would like to thank:

- **Ole Caprani**, for being our supervisor. From the first day you introduced us to the embedded systems, we knew this subject would be time-consuming and source of much satisfaction. You were really enthusiastic with both of our projects, we did our best to show you unexpected things and we hope what we achieved will bring fuel for thoughts. Thank you for taking time for us, answering our question and sharing good moments ***bok bok bok bok… Zzzzz…***;

- **Marianne Dammand Iversen**, for having written an article for *Insight@Katrinebjerg* about our project;

- **All the people from our corridor**, for enduring the noise we made during some of our experiments;

- **Jakob Fredslund**, for giving us inspiration thanks to his PhD dissertation *Simplicity Applied in Projects Involving Embodied, Autonomous Robots*;

- **Morfars**, for the advertisement and being there when we needed spare parts for the drone;

- **Parrot**, for quoting our work as a "great robotics project" in their news feed…

- **X-Ray (energy drink), tea and coffee**, for keeping us awaken during a lot of nights. We could not have met some deadlines without them.

- **Our families**, for supporting us in every possible way. Driving 2000km round-trip and being always there when we needed it is something that is worth mentioning here.

- **Anne** and **Clara**, for being the first to test our new robots, provide user experiment feedback, and -last but not least- spending plenty of Thursday afternoons playing card games with us.

Guillaume Depoyant
& Michaël Ludmann,
Aarhus, August 31st, 2011