THE UNIVERSITY *of* EDINBURGH

Robotics: Science and Systems

Practicals - Final Report

Design, mechanics, and logic of an autonomous robot made out of LEGO parts

Henrique Manuel Martins Ferrolho
s1683857 - henrique.ferrolho@gmail.com

November 21, 2016

Design, mechanics, and logic of an autonomous robot made out of LEGO part

Robotics: Science and Systems - Practicals Final Report

Henrique Ferrolho

Abstract

This report describes the building process of a robot made out of LEGO and electronic parts. The robot was capable of autonomously navigating an arena known *a priori*, finding special resources spread throughout the arena, and delivering those resources to specific delivery points.
The robot used three light sensors and a camera to locate the special resources on the arena, as well as two IR sensors, a sonar, and a hall effect sensor to navigate through the arena avoiding obstacles and keeping track of the total distance traveled.
The source code abstracts and interfaces all the sensors, and implements a State Machine to complete all the components of the global task.
The robot was tested in 10 time trials, having successfully delivered 2 cubes in 7 out of those trials. The average run-time of those 7 trials was approximately 6 minutes.
The major downside of the robot was that it was not able to deliver a cube from one room to a base on another room. This was not a physical layout flaw, but a downside on the programming of the robot.

Contents

1 Introduction

The presented task was to build a robot capable of autonomously navigating through an arena, avoiding obstacles and walls, looking for three unique textured cardboard cubes, and delivering each cube to its respective base - each of them belonging to one and only one base. The entire task should be fulfilled in under 5 minutes.
The arena layout was static, and contained two bases marked with a black rectangle on the floor. The assignment of each cube resource to its base can be easily configured in the robot's source code - as it should, since such assignment were to be changed just before the final practical demonstrations.

The entire project was divided into two major milestones. Each milestone was approached by dividing it into other small sub-tasks. Each sub-task was then completed in a sequential manner, from the most simple to the most daunting. The sub-tasks required for the first major milestone had a bigger priority, and thus were approached first.

It is important to note that the task was not approached by building the entire physical layout of the robot first, and only then programming it. On the contrary, the robot was built iteratively: the physical layout got slightly tweaked each time a sub-task required it. The final physical layout of the robot still used some elements of the initial design, but overall it changed completely.
The same thing can be said about the source code: the architecture was iteratively adapted to integrate new modules. Each module was separately tested before being integrated into the main program.
There were times when the source started to have code smells which required a minor (and some times a major) refactoring. Close to the end of the project a major refactoring had to be done in order to implement a proper State Machine. In retrospective, the State Machine should have been one of the first things to have been outlined and implemented - it would have saved a considerable amount of time throughout all the development process.

2 Methods

2.1 Essential components

This section contains lists of all the components required by the final robot.

Power and logic boards:

- Fit PC
- Power board
- DC motor control board
- Servo control board

Sensors and camera:

- Light sensors (x3)
- IR sensors (x2)
- Camera
- Sonar
- Hall effect sensor

Actuators and battery:

- DC motors (x2)
- Battery

Up until the first major milestone, the robot also had two whisker sensors, but they have since been removed. Their initial purpose and the reason why they were removed are documented on section Sensing: Whiskers

2.2 Physical architecture

The entire structure of the robot is made out of LEGO parts. The robot has two large rubber LEGO wheels on the front, and one LEGO steel ball caster on the back. The robot is driven by the two large wheels on the front, and the caster wheel is just for support.
This design was preferred from the beginning because it allows the robot to rotate on itself, i.e., it can rotate any amount of degrees without actually moving relatively to the arena.

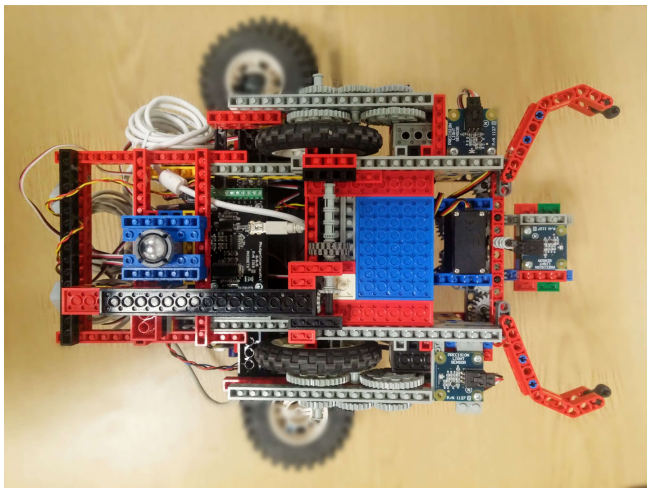


Figure 1: Bottom view of the robot. The caster wheel can be seen on the left of the image, and the two large rubber wheels on the center. The other wheels on the blurry background are not part of the robot and were used just to support the upside down robot while this shot was taken.

To take advantage of the quite heavy battery of the robot, it was placed as close to the front wheels as possible. Doing this added pressure on the tyres of the wheels, increasing the friction between them and the ground of the arena - which is considerably slippy from all the dust.

The power boards and the Fit PC were mounted on the back of the robot, between the two front wheels and the caster wheel. Doing this, and placing the battery close to the front wheels ensured that the center of mass would stay between all the wheels, approximately evenly distributed by them. This was a crucial element of the robot's design.

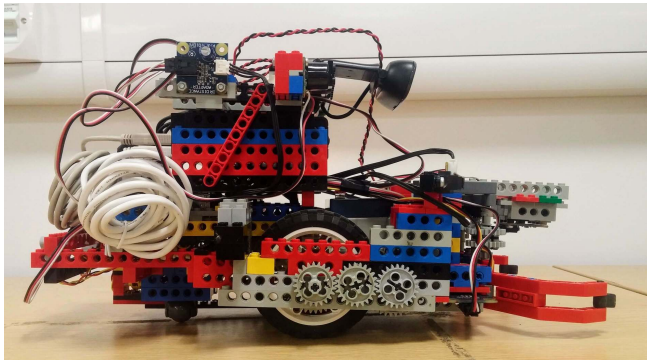
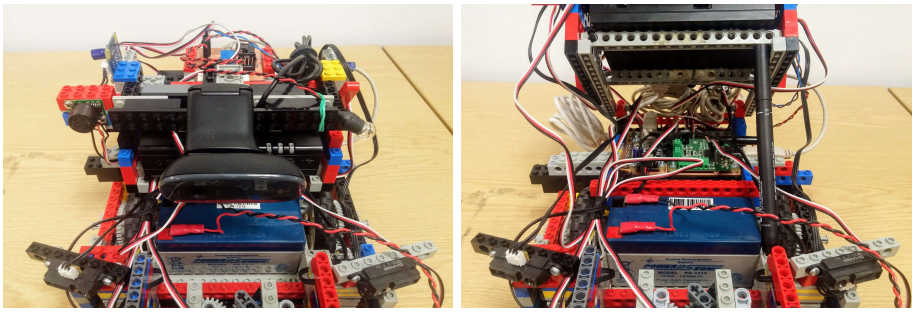


Figure 2: View of the right side of the robot.

It was decided from the very start that the main Power Board should be easily accessible to plug the AC adaptor and to turn On/Off the DC motor power board. For that reason, the Power Board was placed on the very top of the robot, just behind the camera.

The need to constantly plug and unplug sensors to the Phidgets Boards made it clear that there should be an easy way to get to those sensor slots. This was even more important with the constant battery switching when they ran out of power. To solve this problem, a *Hop On - Hop Off* with hinges was devised. To gain access to the boards and to the battery slot, one needed only to lift the hop.



(a) Hop on (b) Hop off

Figure 3: Lifting the hop reveals the Phidgets boards and enables easy access to the battery slot.

2.3 Base detector and gripper

The gripper had to be placed on the front in order to catch the cubes when the robot drove to them. The precision servo motor was used to open or close the gripper handles.

The tips of the gripper handles had a rubber LEGO part to prevent scenarios when the robot closed the gripper and the cube resource was not completely embraced by the handles. Having the rubber tips granted that the cube wouls still be secure in such situations.

One light sensor was mounted facing down on top of the gripper dock, at such a height that a cube would fit just underneath it. Since the cubes are textured on the sides, but are solid black on the top and bottom, placing the light sensor in such position made it very easy for the robot to detect whenever there was a cube on its gripper dock or not.

To complement the gripper, two more light sensors were placed at the front of the robot, one in each side, just hovering the floor and facing down.
These constituted the base detector for the black rectangular bases on the arena, and they functioned very similarly to the cube detector on the gripper: they measured the amount of light reflected by the gray floor of the arena, and triggered whenever black was being sensed instead of gray - this meant that that

sensor was on one of the bases.
When the robot moved forward, carrying a cube and suddenly both light sensors from the base detector got triggered, it meant the robot had arrived to the base and it could release the cube to deliver it.

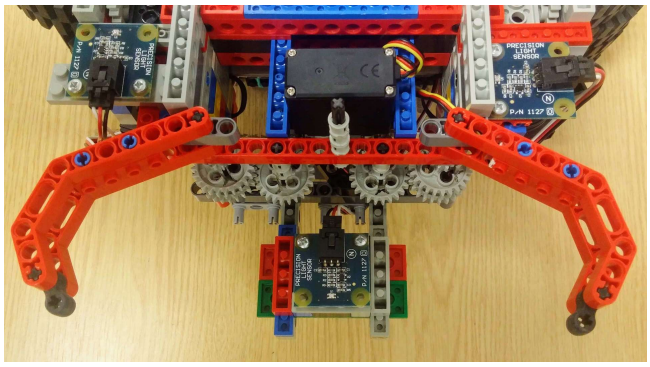


Figure 4: The base detector light sensors can be seen on the left and right top corners of this image. On the center, there is the precision servo motor, the gear train to open or close the gripper, and finally, on the middle bottom, the cube detection light sensor.

2.4 Actuators and gearing

The robot used two DC motors, each to drive its own wheel. However, the wheels were not directly connected to the motors. To have the best power/speed ratio, the power output of the motor had to be geared down with a gear train.
The DC motor output axle had a *16-teeth* gear attached to it, which was connected to a *40-teeth* gear. The rest of the gear train was just to transfer the power to a parallel axle, where the wheels were mounted to.

The total final gear ratio was **1:2.5**, which means the speed output by the DC motor was **decreased 2.5 times**, and the torque was **increased 2.5 times**. Furthermore, this gearing train made the wheels rotate 0.4 times per each revolution of the DC driver motor.

2.5 Sensing

This section describes the sensors used by the robot to collect inputs from its surroundings.

2.5.1 IR and Sonar

The robot used two IR sensors and a Sonar for its collision avoidance system and navigating on the arena.

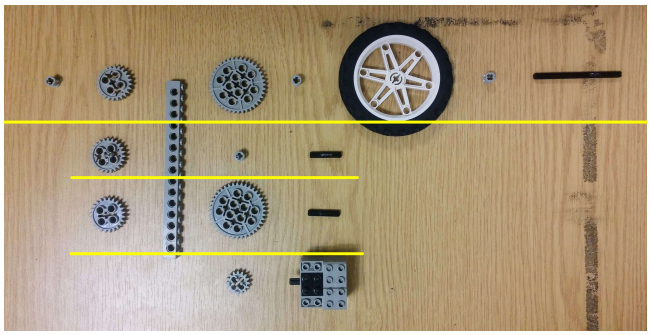


Figure 5: All the components required to assemble the gear train of the robot, from the DC motor to the wheel. The yellow lines separate each stack of elements on an axle, e.g., all the LEGO parts above the first yellow line should be stacked to the black axle on the right, following the order by which they are layed out.

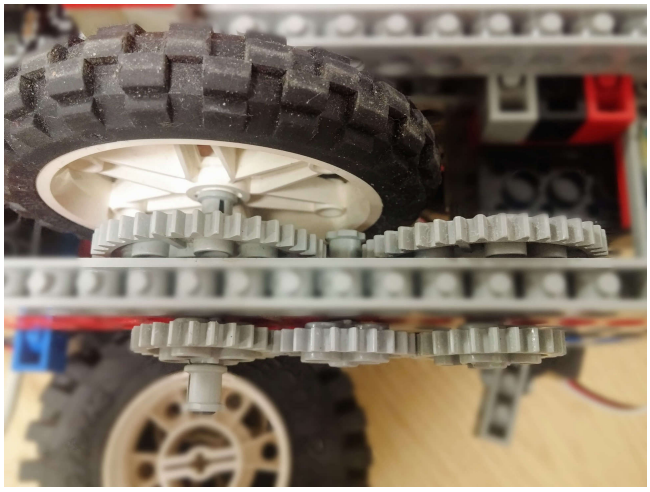


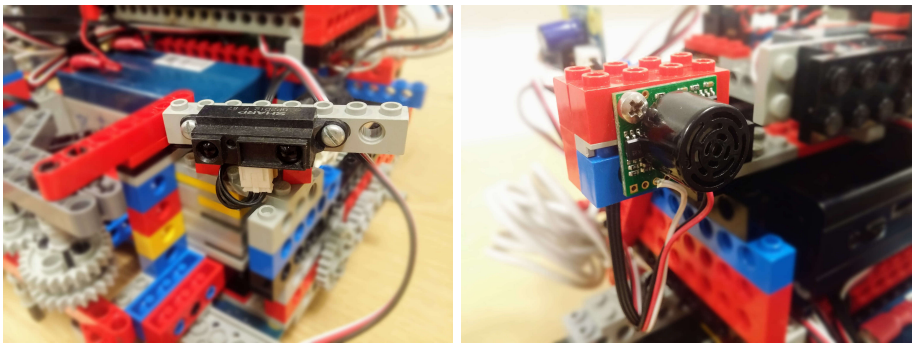
Figure 6: The assembled version of the gear train on the robot's main structure.

The two IR sensors were mounted on top of the DC motors, at the front of the robot, pointing mostly forward, but each at an angle of approximately 14° outwards.

The Sonar was placed at the top of the robot, right next to the camera.

2.5.2 Whiskers

Up until the first major milestone the robot made use of two whisker sensors. Their purpose was to make up for the blind angle caused by the two IR sensors pointing slightly outwards. They were essential for the demonstration of the first major milestone because the robot navigated the arena randomly, and there was the possibility of it driving towards an edge of a wall without the IR sensors to get triggered. The whiskers would be triggered in such cases and would ensure the robot could back up and re-plan its route.



(a) Left IR sensor (b) Sonar

Figure 7: (a) shows where the left IR sensor was placed, and at what angle. (b) shows the sonar placement, facing forwards.

The whiskers have since been removed because the final robot did not have a *reactive* behaviour only, it also had some *planning* - and that classifies it as a *hybrid* behaviour. Such planning ensured that the robot would never navigate the arena in a way that it could approach a wall edge without the IR sensors to be triggered. This turned the whisker sensors obsolete and ultimately led to their removal.

2.5.3 Camera

The camera attached to the top of the robot was used to scan the arena for cube resources, approaching those cubes, and identifying them.

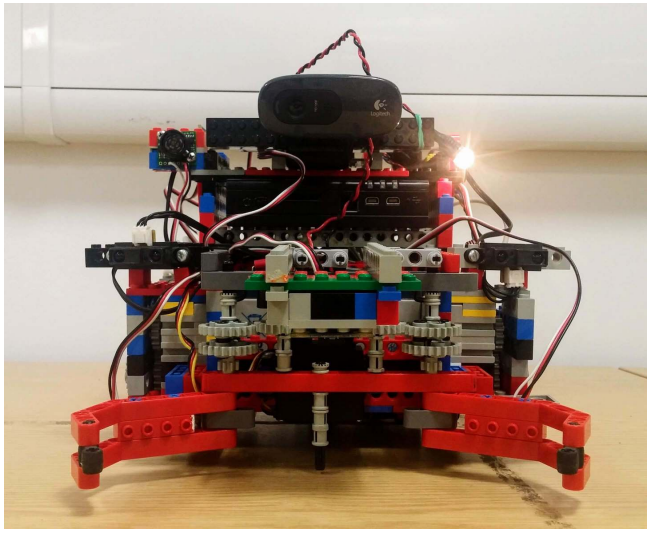
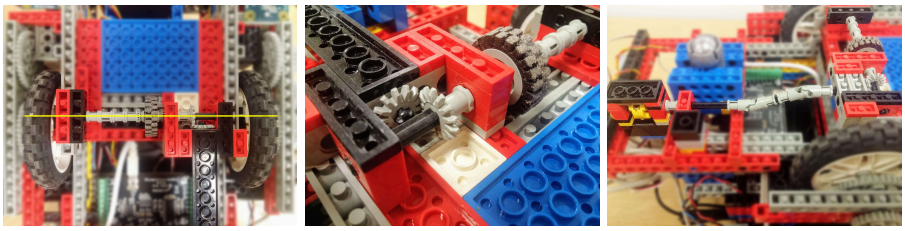


Figure 8: Front view of the robot featuring the camera on the top, the Sonar to its left, and the IR sensors just below them, pointing slightly outwards.

2.5.4 Hall effect sensor

A hall effect sensor was used for *odometry* to keep track of the distance travelled by the robot.

A small pivot wheel was placed in-between the two large wheels that drive the robot, aligned with their axle. The axle on which this pivot wheel is attached transfers its rotation to a perpendicular axle, which rotates below the robot, just like a shaft, and in its turn inputs its rotation to the hall effect sensor.



(a) Pivot wheel placement (b) Axle transfer (c) The whole mechanism

Figure 9: (a) shows the placement and alignment of the pivot wheel. (b) shows gearing to transfer the rotation from the pivot wheel axle to the shaft. (c) shows the entire mechanism, from the pivot wheel to the hall effect sensor.

2.6 Logical architecture

All the source code of the robot was programmed in **Python 2.7**, and has been made publicly accessible on its entirety at https://github.com/ferrolho/uoe-rss.

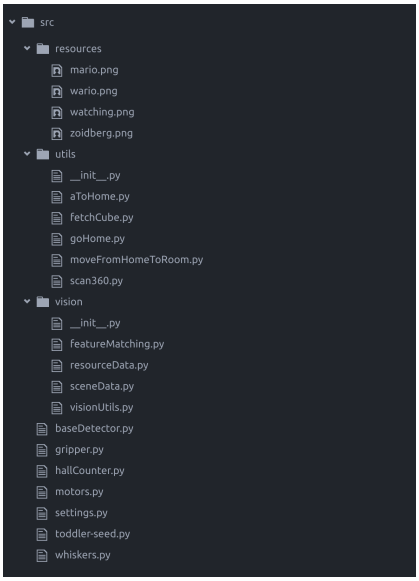


Figure 10: The source code file structure and modules.

2.6.1 The `src/` root folder

The `src/` folder is the root of the project. It contains two modules: **utils** and **vision**, the **resources** folder, and other source code files.

The `resources/` folder contains all the four different textures that the robot could find while navigating through the arena.

`settings.py` is where the main configurations for the robot are, including the assignment of each resource to one of the bases of the arena.

`baseDetector.py` is the the class that abstracts and interfaces the two light sensors responsible to detect the two black rectangular bases of the arena.

`gripper.py` is the class that abstracts and interfaces the gripper. Convenient methods to close and open the gripper exist, as well as methods to check if there is a cube on the gripper.

`hallCounter.py` is the class that processes the measurements of the hall effect sensor. It keeps track of the total distance travelled by the robot. Also contains a handy method to set a timer which goes off after the robot as travelled the distance set by the `setTimer()` function.

`motors.py` is the class that interfaces the motors. Multiple simple behaviors have been abstracted to make the development easy. For example, to make the robot move forward, one needs only to call `self.moveForward()`.

`whiskers.py` is the class that abstracts the whiskers.

`toddler-seed.py` contains the `Toddler` class, and is the main program, and the starting point of the robot control flow. The class implements a *Finite State Machine*, which can be summed up in the following keypoints:

1. Scan 360 for a room
2. Move into that room
3. Search for and fetch the cube
4. Deliver cube to the respective base
5. Return home

2.6.2 The `utils/` module

The `utils/` module contains all the high-level source code for the fundamental procedures to solve the task.

`scan360.py` is a high-level procedure which implements a simpler state machine to solve the first problem of the task, i.e., when the robot was placed on the center of the arena, facing a random direction, this procedure was the one responsible to turn the robot until it was facing the room entrance of interest.

`moveFromHomeToRoom.py` this procedure is the one responsible to plan the robot route from home (the center of the arena) to the room of interest. It assumes

that the 360 scan had been ran previously, and as such, that the robot was already facing the room it wanted to move into.

`fetchCube.py` is the routine responsible for scanning the room for a cube resource, and when it spots a cube, approaching it and locking it on the robot's gripper.

`goHome.py` is the high-level abstraction to return home assuming the robot had just delivered a cube to the base in room B.

`aToHome.py` is pretty much the same as `goHome.py`, the difference is on the planning routine to go home for cases when it is assumed the robot just delivered a cube to the base on room A.

2.6.3 The `vision/` module

The `vision/` module contains all the source code required to process and extract important information from the images taken by the robot's camera.

`sceneData.py` and `resourceData.py` are two simple and very similar classes that represent the information stored on a frame, i.e., a frame keypoints and descriptors, which are required to run the SIFT algorithm.

`featureMatching.py` is the source file which contains the function to run the SIFT algorithm between two images.

`visionUtils.py` is the most important class of the module, and creates a level of abstraction for sub-tasks like scanning for cartoons on an image, and looking for a cube in the distance.

3 Results

3.1 360 scan

When the robot was placed on the center of the arena, it could always find the entrance to the room it wanted to navigate to, no matter its initial orientation.

When the robot was started, during the initial 360 scan, it turned on itself clockwise, using the IR sensors to sense its surroundings.
The procedure responsible for the 360 scan waited for both sensors to be higher than a threshold of 180, and identified that as being the only wall that could be perpendicular to the direction the robot was facing. After that, it counted the number of gaps with the left sensor and stopped on the gap of the room of interest.

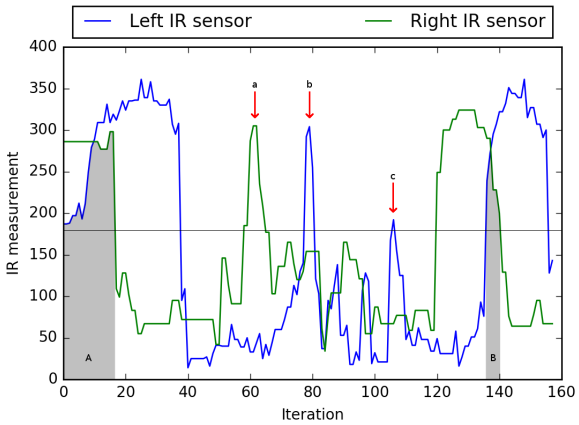


Figure 11: Samples of the IR sensors while performing a 360 scan at home (the center of the arena). (A) and (B) are the moments when both sensors are triggered past the threshold, i.e., the robot is perpendicularly facing the wall of the center of the arena. (a) and (b) are the moments the right and left sensor pass by the edge of the wall between the entrance for room B and room A, respectively. (c) is when the left sensor passes by the edge of the wall between room A and room C (the smallest of the rooms).

3.2 Fetching cube resources

The robot processed the camera frames by making two copies and cropping each of them: the first would be cropped vertically, removing 2/7 of the width on each side; the second one would be cropped by removing the bottom half of the frame.
The first copy would be used to run the Feature Matching (SIFT) algorithm, in order to identify the textures on the resources.
The second one would be the long range distance view of the robot, by filtering

all colors outside the HSV color space range present in the textures.
Cropping these images increased the speed of their post-processing.

The robot was always able to classify the resources at a distance below 0.5 m. It was also always able to identify an object of interest, i.e., a cube resource, up to 1.5 m.

The robot used the coordinates of the contours of the cube on the processed images to calculate its position with respect to the cube. According to that calculation, the robot turned on itself to center the cube resource on its field of view, and then advanced towards the cube 4 hall units.

Occasionally, the cube sensor on the gripper missed the cube, making it believe it did not catch the cube. In such situations, the robot was programmed to back up a little, and re-run the normal procedure.
3 out of 10 test runs, the robot had the cube on its gripper but thought it did not. In those cases, the robot backed up and retried to catch the cube.
In 1 out of those 3 runs, the robot drifted so much due to the imprecision of its motors that it was not able to get the cube at all.

3.3 Homing after delivering a cube

The strategy of the robot was to go to the biggest room first, find the cube, deliver the cube to the base on that room, and then return home.
To return home, the robot made use of odometry and its sonar to follow the preplanned route from the base to the center of the arena.

Occasionally, when the battery was about to die, the difference in drag between the motors increased, in its turn increasing the drift of the robot as it moved through the arena. Unsurprisingly, in such situation the robot accumulated enough error to not manage to get back home.
However, when the conditions were acceptable, i.e., the battery was not dying, and the floor did not have debris, the robot was always able to return home after delivering a cube to the base in room B (the biggest room).

The same did not hold true for homing after delivering a cube to base in room A. The approach used to get the robot back did not work all the times, and needed to be improved for greater reliability.
Out of 20 test runs, the robot did not manage to get home in 5 of those runs.
Furthermore, since the robot relied on this step to get to the third and last cube, if the robot did manage to return home but did it in a way that it got blocked while doing the last 360 scan, the robot was not able to get the last cube. And this is the reason for the robot to fulfill the complete task very few times.
On 9 out of the remaining test runs where the robot managed to get home, it did get stuck.

4 Discussion

The most successful aspect of the robot was the ability to consistently orientate itself on the center of the arena, and move into a room of interest.
The final physical layout of the robot turned out very robust and flexible.

The robot was able to deliver two of the cubes most of the times. Occasionally, it was able to deliver all the three cubes. However, it was not able to deliver a cube from room A to room B, and vice versa.

There is room for improvement regarding the vision of the robot: the algorithm could be enhanced and sped up.
Also, the procedure for returning home from room A, and the procedure to go from room A to room C could be tweaked even further in order for the robot to be able to pick the third cube with a greater success rate.

5 Sources

Python 2.7 Docs
https://docs.python.org/2.7/

OpenCV 3.0.0 Docs
http://docs.opencv.org/3.0.0/index.html

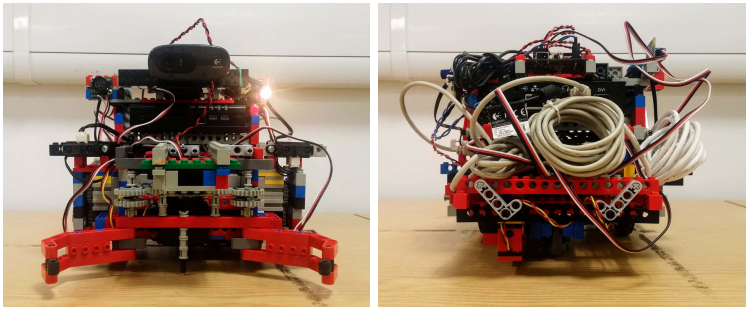
The Art of LEGO Design
http://www.cs.tufts.edu/comp/150IR/artoflego.pdf

LEGO Design
https://www.clear.rice.edu/elec201/Book/legos.html

Gears, Pulleys, Wheels, and Tires
http://www.ecst.csuchico.edu/~juliano/csci224/Slides/03%20-%20Gears%20Pulleys%20Wheels%20Tires.pdf

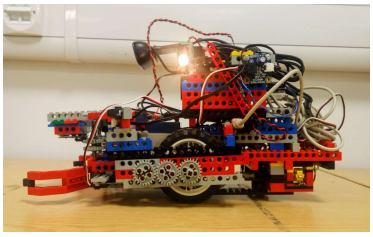
LEGO Gear Ratio Calculator
http://gears.sariel.pl/

A Appendix

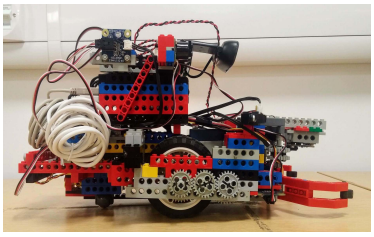


(a) Front view

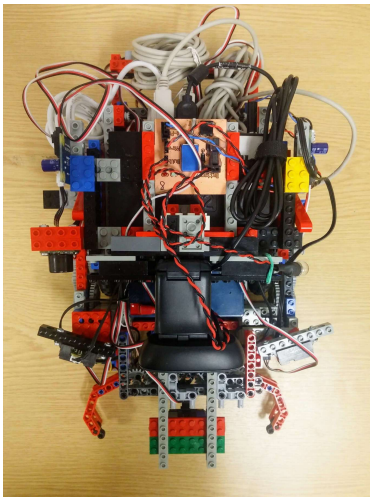
(b) Back view



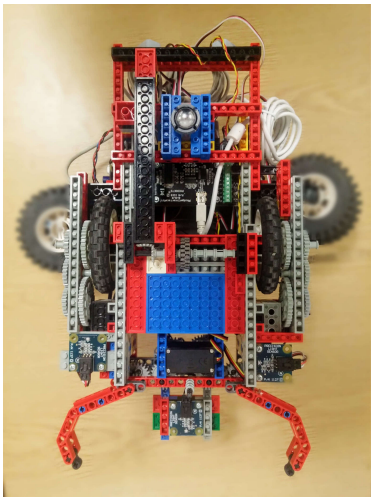
(c) Left side view



(d) Right side view

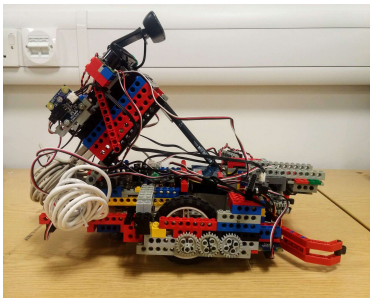


(e) Top view

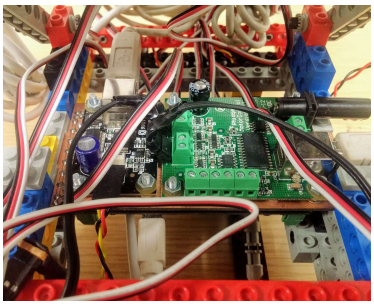


(f) Bottom view

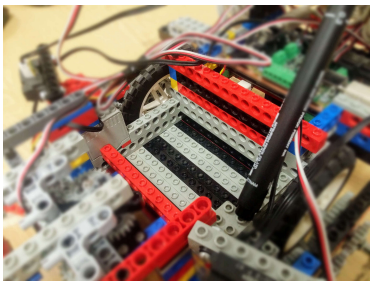
Figure 12: Different viewpoints of the robot.



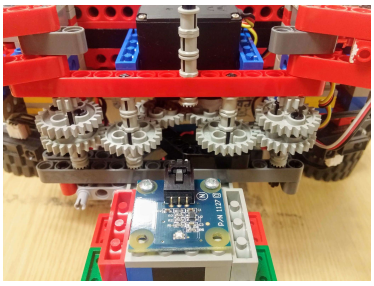
(a) Hop off - side view



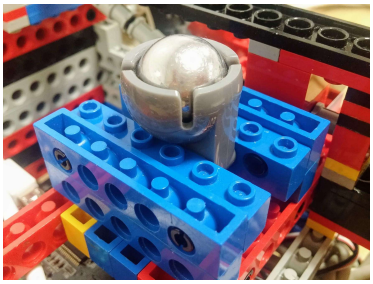
(b) Phidgets boards



(c) Battery slot



(d) Gripper gear train



(e) Steel ball caster

Figure 13: More images of robot details. One can see that when the hop is off (b) and (c) become easily accessible.