Part 1: Detailed System Design Document

1.1.0 Physical structure

Provided hardware

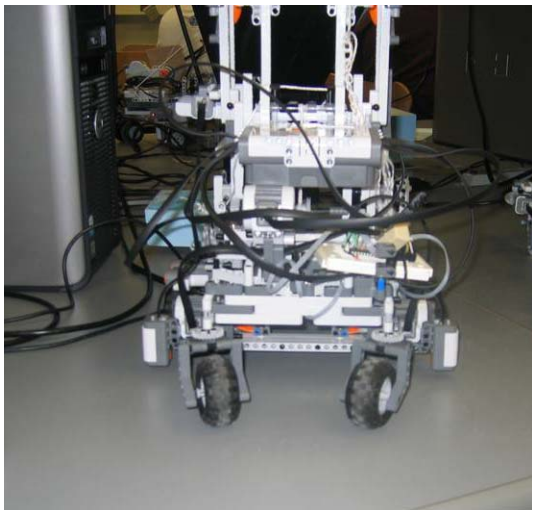
The first step of the design process was to decide upon a physical structure that would allow us to solve the given problem. Before creating this structure it was necessary to take inventory of the existing technology (Lego pieces) that would be used in the construction phase. With these limitations in mind, it was then necessary to decide what exactly our structure would do. This was accomplished by answering some key questions:

- How would the robot move?
- Would it lift boxes or drag them?
- How would the robot lift the boxes?
- Would it stack them on itself? Or only carry one box at a time?
- How would it be able to stack on itself?
- Are there any required pieces that will not be included in the lego set?
- How many bricks will be needed?

After much deliberation, preliminary answers to these questions were obtained. These answers, as well as an examination of the process leading to the final product, will now be explored.

The structure designs described in this section were completed using solely the hardware provided in the Lego kits. Some extra added parts include the use of elastic band in order to allow the side bars, used to orient wayward boxes, to be able to have some ability to rotate and not snap

1.1.1 Transportation

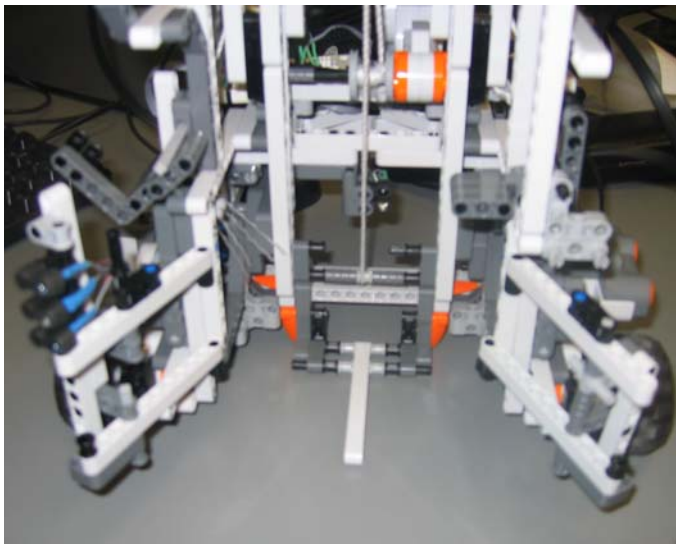


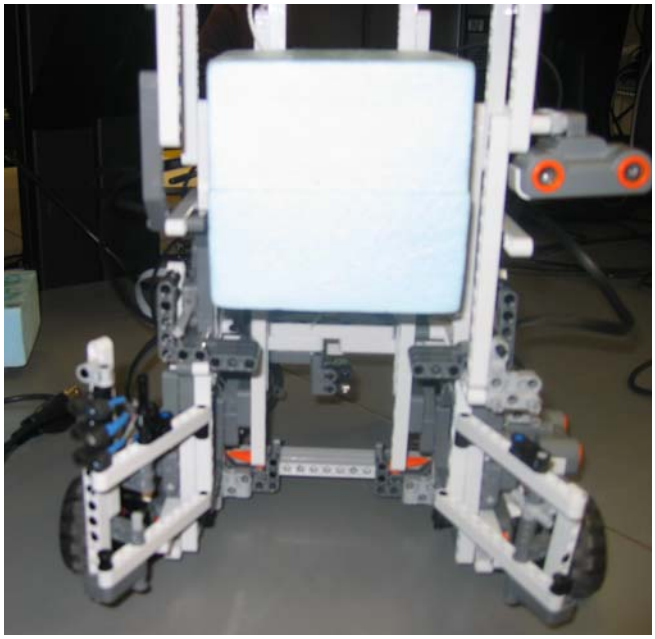
While not differing vastly from the original design of the tribot, some changes were made to accommodate the increased overall size of the design. Specifically the major difference is the use of two full-sized wheels as caster wheels opposed to the previous use of a small plastic piece of Lego. The benefits of this design are two fold. It allows an even distribution of weight to all four wheels, as well as supporting the increased weight of the robot (two wheels would not suffice). Secondly, these larger caster wheels were designed in a very efficient manner. They reduce the amount of slippage

when compared to the original design, decrease the turning radius, and are less likely to be caught in small bumps on the driving surface.

The control of the wheels is once again similar to the original design. One motor controls one wheel each left wheel controlled by the left motor and right wheel controlled by the right motor. The caster wheels are not connected to any motors and operate in a passive manner.

1.1.2 Lifting Mechanism





It was decided that the robot would lift the boxes instead of dragging them. The reasons for this rely on both the point structure for the competition coupled with the knowledge that the driving surface would not be perfectly smooth.

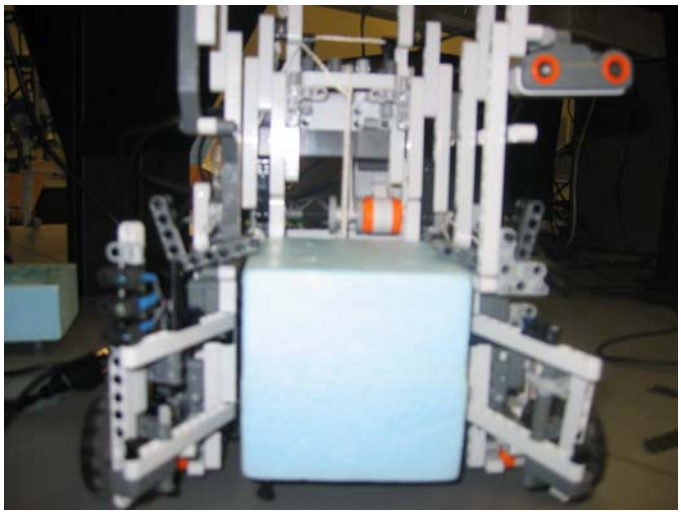
The design of the actual lifting mechanism is relatively simple. First, to ensure that the robot is able to capture as many boxes as possible regardless of their orientation, we have implement side bars that will direct wayward boxes towards the center of the lift.

The lift itself is comprised of only one fork but remains capable of easily lifting at least three boxes at a time. This design was chosen since it was the simplest lifting mechanism that remains able to complete its given task.

The fork is attached to a motor via a string. This system acts like a simple wench. Once again, this design was the simplest and most efficient available, requiring only one motor to operate. The lift moves up and down along tracks to ensure that there is no horizontal translation during lifting and to negate any tilting effects of the fork.

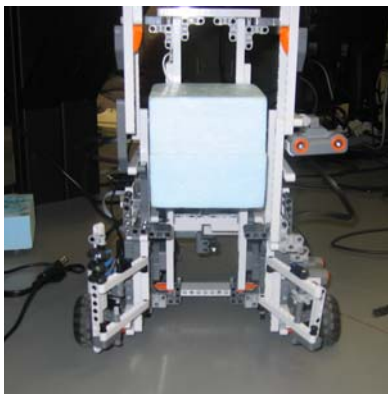
The motor control of the lift is highly intelligent, thanks to clever programming. Once a box is detected the lift begins to move upwards at full power until it reaches a predetermined height. Once it reaches this height, it slows down and the speed is constantly monitored. If the current speed decreases by 75%, since due to an increase in tension, the lift has been raised sufficiently high and can now return to its original position. This design combats and effects of slack created by over-use of the string since it can become stretched over time.

1.1.3 Stacking Mechanism



In order to stack the boxes before reaching the stacking area it would be necessary to stack the boxes on the robot itself. In order to accomplish this task there must be some system in place that will be able to hold multiple boxes in place. In our design we have provided to transport 3 boxes at a time. The process for stacking the boxes is as follows. As the first box is lifted it activates the two passive latches, which will hold the stacked boxes in place, and forces them open. The design of these latches has evolved over time. At first they were simple straight bars with the pivot located near the tracks. This design unfortunately was not very efficient since it required a large amount of movement to fully open the latches causing boxes to become stuck occasionally.

A new design was agreed upon, implementing L-shaped latches to allow them to open much quicker than previously; furthermore, plastic tips were added to an extension of the latches to reduce the movement of the boxes on the latches. A final adjustment to the latches was to put counter-weights on the outside edges combating the adverse effect when a box was completely on side of the lift.



Once the fork reaches its maximum height, it causes the latches to close. This closing process is accomplished by the actual movement of the fork, as it approaches its maximal height the top of the fork pushes against a rope that connects the two latches. As this rope is pushed upwards the latches close, allowing the box to be moved back down and held in place. The first box lifted is lifted to the maximum height then placed on the latches. The second box pushes the first box up and then they both become stacked on the latches. Finally the third box is only lifted up slightly. When the box need to be

unstacked they are all lifted halfway up allowing the latches to open, and then they are all placed in the appropriate spot.

At first a door was used in order to maintain the stability of the stack. In order to operate this door effectively it would have been necessary to control it with a motor, or at the least couple it to the function of a different motor, since the door needs to be open whilst stacking and unstacking and closed during lifts. This design was replaced with a passive mechanism that holds the stack of boxes in place (shown in the closest image).

Additional Hardware: Proprietary Sensors

1.1.4 Motivation

Our previous experience with the Lego NXT kit let us become familiar with the capabilities of the Lego sensors. The light sensor, in "reflected light" mode, is good for detecting the presence of an object at distances less than about 4 cm. The ultrasonic sensor is good for sensing and measuring the distance to objects that are between 30 and 250 cm away and favorably oriented. What we really needed for our robot, however, was a sensor that could reliably detect the presence of objects in the range from 0 to 20 cm.

1.1.5 Sensor concept

The main problem with the ultrasonic sensor is that sound wave reflection is highly dependent on material and, more importantly, on the orientation of the surface. Most surfaces reflect sound very normally if at all, so a reflection will only be easily observed if the emitter and detector have a "line-of-sight" which is perpendicular to the surface. With electromagnetic radiation like visible light, on the other hand, most surfaces do not reflect normally (only very shiny surfaces like mirrors reflect light normally). So, if we use light reflection instead of sound reflection for detecting objects, we don't have to worry about the orientation of the surfaces we are detecting.

When using the light sensor in "reflected light" mode, light readings increase when an object approaches because more of the emitted red light can reflect off the object and be detected by the phototransistor. However, the phototransistor also reacts to changes in ambient light, so when light readings change, it is hard to determine what caused the change. One way to eliminate this ambiguity is to compare light readings taken with the emitter illuminated to light readings taken with the emitter extinguished. Since ambient light introduces the same offset (assuming sensor linearity) whether the emitter is on or off, its effect can be eliminated by subtracting the emitter-off reading from the emitter-on reading.

We decided to build a sensor similar to Lego's light sensor. The most readily-available phototransistors were found to be most sensitive to light at the infrared end of the spectrum, so we chose to use infrared light emitting diodes (IR LEDs) as emitters.

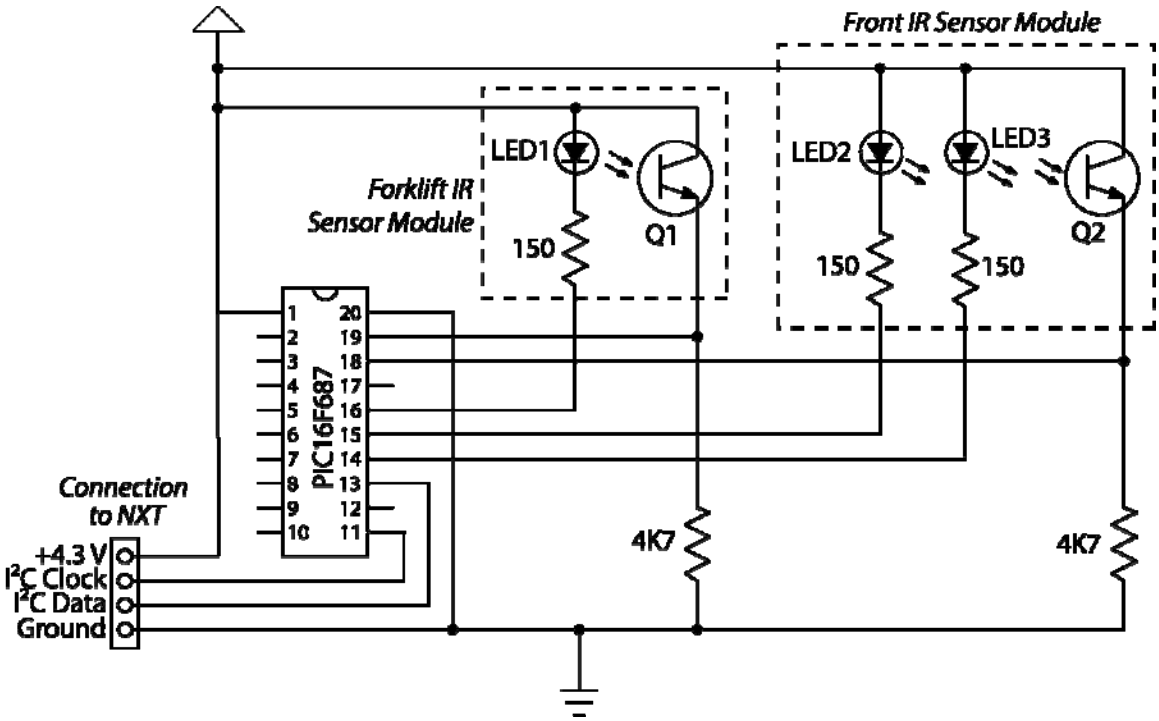
1.1.6 Sensor interface

The NXT interfaces with sensors in one of two different ways. It can sample an analog voltage, which is useful for reading very simple sensors which provide a varying resistance as a function of physical stimulus. Or, the NXT can interface digitally over a serial bus. The NXT's serial bus implementation adheres to the I^2C standard. There are a few advantages to using the serial bus: multiple devices can be connected to the same input port without needing any multiplexing electronics; and, for each device, different variables can be requested by the NXT. A disadvantage is that a microcontroller or dedicated I^2C interfacing chip is required.

Our robot was already going to be using at least one ultrasonic sensor, which is a digital sensor, so adding an additional digital sensor device to our robot would require no additional input ports. This was seen as a major advantage, and so we decided to devote resources towards using a microcontroller to interface our infrared proximity sensors with the NXT. We chose the PIC16F687 microcontroller because we had experience with 8-bit PICmicro microcontrollers, and this particular model had built-in support for I^2C, reducing hardware and software requirements.

1.1.7 Circuit Design

The circuit for our device is relatively simple. The on-board components include the microcontroller and a few resistors. The off-board components include one detector-emitter pair (three wires), and a unit comprising one detector and two emitters (four wires).

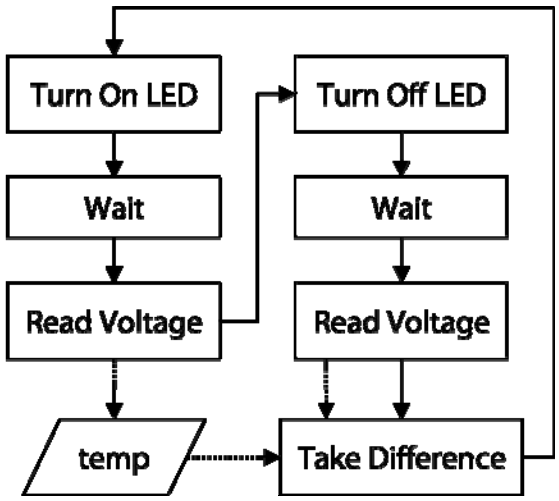


The two off-board modules (enclosed in dashed lines) are wired in a common-anode/collector configuration. The LEDs are each connected to a GPIO pin on the microcontroller through a current-limiting 150 ohm resistor. The phototransistors Q1 and Q2 allow current to flow as a function of incident light. Pins 18 and 19 on the microcontroller are high-impedance analog inputs; any current flowing through a phototransistor must also flow through its respective 4.7 kohm resistor. This resistor acts as a transresistance amplifier, creating a voltage at the analog input pin which is proportional to the current flowing through the phototransistor.

1.1.8 Microcontroller Firmware

The starting point for the firmware came from Microchip's Application Note 734. This code provided an example of an I^2C slave device, with the capability of having a master (i.e. the NXT) write values to and read values from any of 7 registers on the slave microcontroller. All of this was taken care of in an Interrupt Service Routine. As far as interfacing was concerned, this was sufficient for our sensor device. All that was left was to construct a main loop which would populate the registers with sensor data.

The main program takes care of turning the infrared LEDs on and off, and reading the analog voltages which are a function of the current through the phototransistors. For each instance where we want to determine the amount of reflection, we follow a simple procedure which lets us mitigate the effects of ambient light.



The delays are necessary because they allow the current in the phototransistor to stabilize (we found that this, rather than the LED illuminating, was the slowest process, taking roughly 250 μs). In the firmware, this procedure is actually executed three times for each infrared emitter, and the resulting difference from each iteration is added to a sum. Summing the values has the same effect as taking an average: it smoothes out noise.

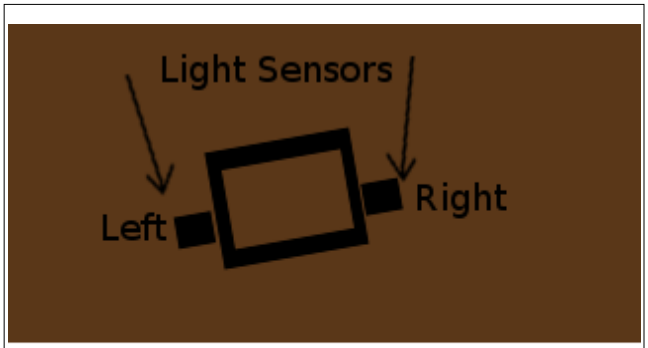
1.2 Software algorithm and implementation

1.2.

Unloading algorithm

The algorithm Megatron used to unload into the stacking area was meant to be as simple as possible, while maintaining high functionality. Since the stacking robot was traveling in a counter-clockwise path around the map, the stacking area would only be entered from one side, so we made adjustments to suit. At the start of the program, measurements are taken of the white surface, and recorded. Other measurements are relative to the differential between them. For the black lines, the difference was around 14 or 15 %, and for the brown paper the difference was from 4-7%. While roving, we took measurements from the light sensors, and waiting for three consecutive readings of brown. From here, we can assume we are likely in the stacking area.

However, the robot is not guaranteed to be straight, and could be angled by as much as 15 degrees (from the wall-following algorithm). From here we must corrected the heading to be perpendicular to the wall, to ensure accurate stacking.

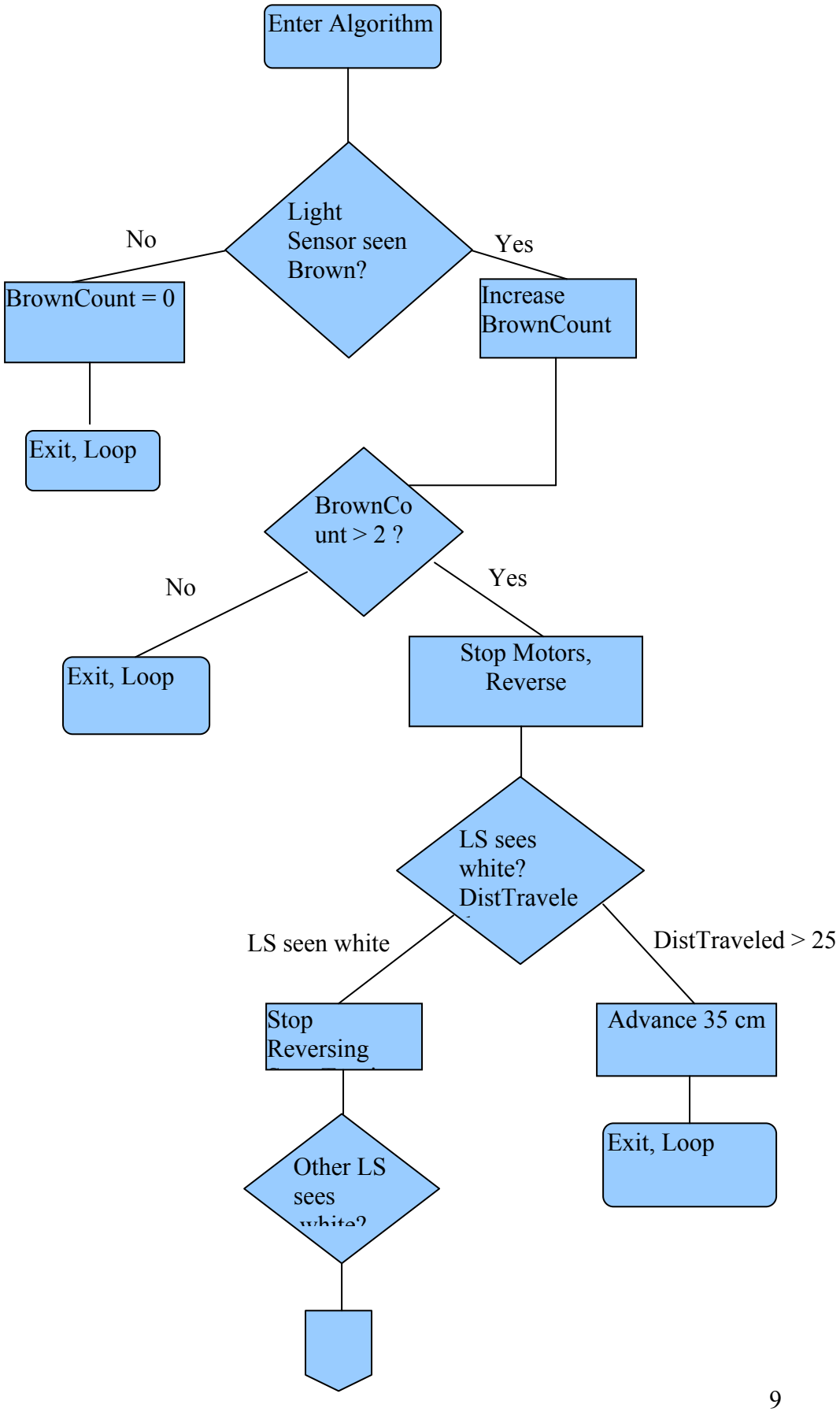


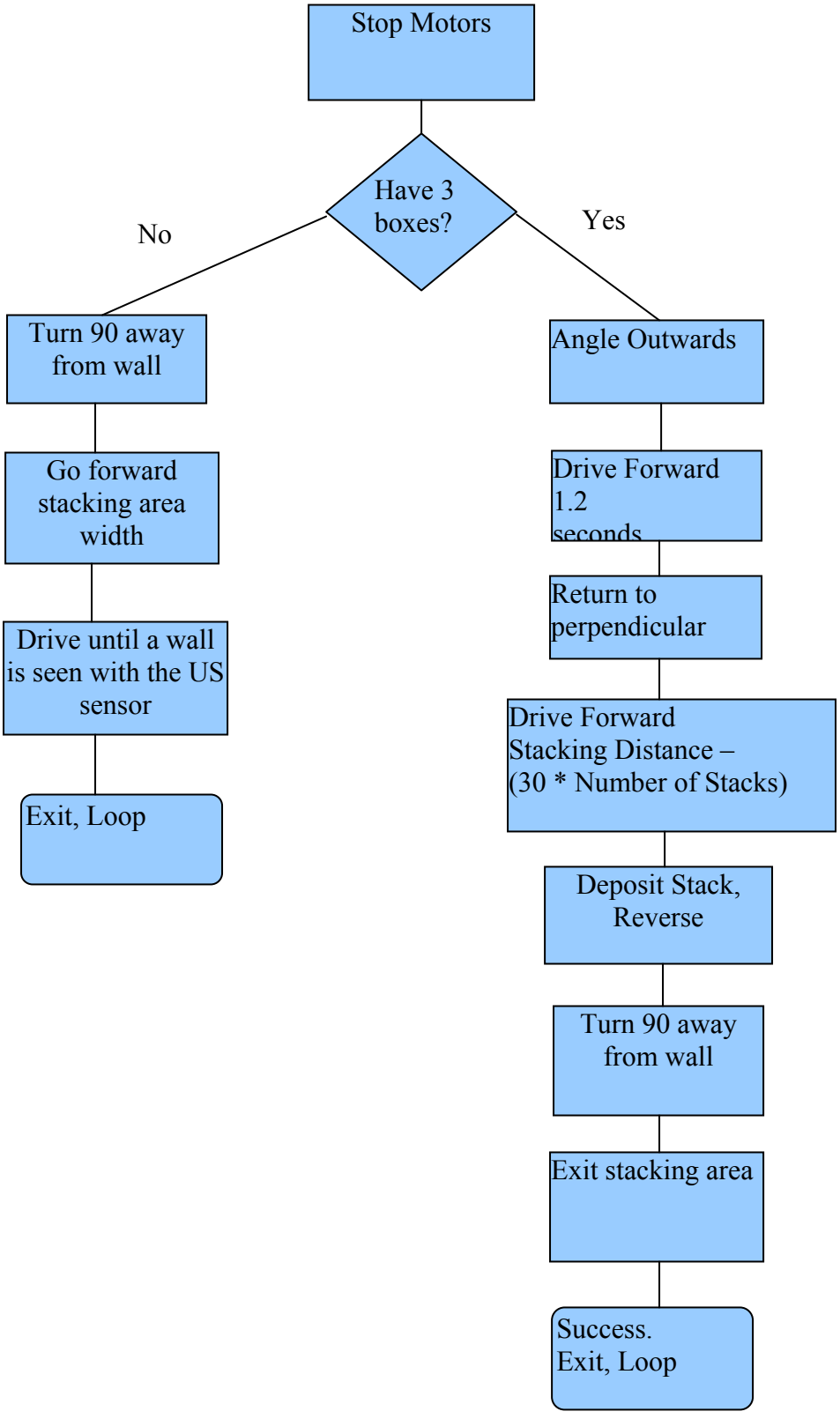
Robot just entering the stacking area

To correct the heading, we first move backwards until one of the sensors registers the white floor. There we stop, and turn the robot until the other light sensor registers white as well. This part of the algorithm may lead to problems if the robots light sensors are in the stacking area from a different angle, or if it becomes impossible to satisfy the condition for reversing, which was to register the reading corresponding to white (e.g. reversing against a wall). To account for this, our software sets a 25 cm limit on which to back up. After this limit has been passed, we assume an error has occured and the robot drives forwards 35 cm to escape the brown stacking area.

After the alignment, the robot is straight and in the stacking area, but we are still a bit too close to the wall. To correct this, the robot simply angles itself outwards, drives forward, then angles itself straight again. From here we can begin the actual stacking process. The length of the stacking area is a known constant, and so the robot drives forward that amount, and releases the boxes by performing a "half lift" with the fork, which releases the clasps that hold the upper 2 boxes and forms a nice stack of 3. Then the fork is lowered to the floor and the robot reverses *slowly* so as not to disturb the inevitably precarious stack. From here we exit the stacking area toward the outside, and find a wall the resume scanning. The next time we enter the stacking area with a stack, the same alignment/adjustment sequence is performed, and then we deposit the second stack 30 cm behind the last one. This is to leave space for other boxes that may have been dragged along into the stacking area.

When the robot enters the stacking area without a full stack of three boxes, Megatron is programmed to avoid the stacking area, for fear of inadvertently knocking over stacks. So, as a precautionary measure, the robot performs the alignment, then a 90 degree turn to exit the stacking area, and then drives forward until it registers a wall, so that it can resume the scanning.


Enter Algorithm
Light Sensor seen Brown?
No
Yes
BrownCount = 0
Increase BrownCount
Exit, Loop
BrownCount > 2 ?
No
Yes
Exit, Loop
Stop Motors, Reverse
LS sees white? DistTravele
LS seen white
DistTraveled > 25
Stop Reversing
Advance 35 cm
Other LS sees white?
Exit, Loop


Stop Motors
Have 3 boxes?
No
Yes
Turn 90 away from wall
Go forward stacking area width
Drive until a wall is seen with the US sensor
Exit, Loop
Angle Outwards
Drive Forward 1.2 seconds
Return to perpendicular
Drive Forward Stacking Distance – (30 * Number of Stacks)
Deposit Stack, Reverse
Turn 90 away from wall
Exit stacking area
Success. Exit, Loop

Part 2: Formal design process

2.1.0 Actual design process

2.1.1 Problem description

The problem proposed is to design and prototype a robot capable of navigating through a 3m x 3m enclosed area and pick up boxes. To motivate vertical stacking, more points have been allocated to 2nd and 3rd layers of boxes.

The robot needs to be self sufficient (no human interaction) and able to navigate and avoid obstacles. The boxes are to be dropped off in a specific unloading area determined by a darker shade of floor material.

Material to be used for this project is restricted to 3 NTX kits and a few specific parts, allowed after management consultation.

2.1.2 Strategy

The first part of the design was to study the documents given to us and brainstorm on potential solutions. After reaching a consensus on how to proceed, we broke up the work into a series of individual systems to be developed and debugged by sub teams.

We worked out two primary design guidelines for our team. First was to adhere to the KISS principle (Keep It Sweet & Simple). This enabled easier communication between the sub teams as well as easier debugging and system integration. Our second focus was to make a design as independent as possible from unknown parameters. By doing so, we were able to debug and enhance performance before competition day and minimized the advent of unpredicted secondary effects of parameter modification.

Throughout the week, we met regularly to update the team on general progress being made. This also enabled us to get feedback from other members on the solutions we proposed as well as different perspectives on how to solve certain of our problems. Since we knew we were going to need extra time towards the end of the project, time reserves were made at earlier stages to allow for last minute requirements (estimated to be greater than weekly allocation).

Once significant progress was made on most systems, we began the systems integration followed by general debugging and performance enhancements (each system had previously been individually debugged and was mostly functional in “stand alone prototype” at this point). We also assigned one member to produce documentation, manage and coordinate our regular reporting.

On the competition day, knowing that a lot more debugging and performance enhancements were to be made on the original track, we assigned the responsibility of public relations to one person and mounted our poster in such a way that half of our work table was hidden from the public.

This had many benefits for the public's impression of our performance as a team compared to others. By hiding the ongoing last minute tweaking, unlike the other teams who were fixing their bugs while answering questions, we gave the impression that the system was flawless and fully functional and that we were ready to compete. To reinforce this impression, we brought a screen to display videos of simulated trials showing the robot in action.

This division ensured that someone was always there to present the final product such that the public never felt like they were disturbing anything by asking questions. It also had the advantage to enable the rest of the team to fully focus on performance enhancement without being disturbed in any way, thereby enhancing their concentration level.

2.1.3 Implementation, performance and improvements

Considering the satisfactory results of the competition, we feel like our design strategy was adequate to solve this particular problem.

Our primary design guidelines were thoroughly followed. Further observation of our work will display simple mechanical systems to replace motor actuators (i.e. latches and door), a compact and solid design and all operations running off a single Brick. We also managed to implement wall following navigation fully independent of the gridlines provided. This also enabled us to find the unloading area no matter where it was placed, as well as avoid any obstacles that may have been placed in the way.

As far as the public relation strategy on the competition day, we hope that we made a good and professional impression throughout our presentation, and allowing most of the team to work on last minute improvements was definitely helpful, as our performance increase from one trial to the other.

The main flaw in our approach was out underestimation of the challenges posed in system's integration. As much as we focused on a simple design, we didn't make enough effort in implementing compatible program flows. This flaw is outlined by the fact that one of our [necessary] system (BOReS, which would have enabled us to pick up boxes out of our path) was fully functional on its own but not part of the final robot.

For future design, a good approach to minimize this issue would be to start with a general algorithm for the whole system such that the sub teams may better coordinate their program flows and variables to allow better synergy. It would also be advisable to assign the "senior programmer" (or more realistically, the student with the most software development experience) to supervise the development of all sub systems with a global vision to ensure the minimization of divergence from a uniform program flow.

We found it also challenging to make deadlines respected by team members. Since everyone is fairly busy and no students have any actual authority over others, it becomes quite challenging to assign deadlines for progress since there are essentially no motivations for respecting them.

As feedback for future classes, it might be interesting to assign a TA for each team, which could serve as that body of authority to make sure that a realistic schedule is in fact followed and that the team has a more global and synergic approach to the design problem.

2.1.4 Time use and allocation

We were able to complete the whole project only 2 hours over time allocation. We also managed (as planned in the beginning) to build sufficient time reserves early in the project to compensate for last minute requirements. Detailed time allocation and usage is presented in **********

2.2.0 Outcome of final product

2.2.1 Performance

Even if the design software is a lot simpler than planned, the robot moves faster and uses less memory. Also, the low ultrasonic sensor was unused in the competition which is one less sensor to read.

- Can stack 3 boxes at a time for one loop
- Stacks correctly about 85% of the time. Usually it does two stacks in about 5 minutes if unexpected cases do not occur due to still existing bugs.
- Will never bump into a stack that it placed previously.
- Handles obstacles very well.

2.2.2. Current Bugs

- No knowledge of the actual number of boxes in the hold of the robot (i.e. if a box falls out by mistake the robot will not notice it and will still think it has the box that just fell)
- Might get on top of a box if the ground surface is not smooth enough while it pushes a box.
- The Robot is only able to stack multiples of 3 boxes(i.e. if it only finds 2 boxes it will never be able to stack them because it can't bring the boxes down from the containment tower)
- Might get stuck in the stacking area due to an open loop code (i.e. the robot doesn't always check for walls/obstacles before driving away from the stacking area)
- It can only catch boxes that are in front or right of the robot in a 5cm radius.
- The robot does not check for boxes that are more than 30cm away from the wall.(i.e. it never checks to its left or in the middle of the maze)
- Cruise speed is too high and compromises performance.
- Can only get to stacking area by following the wall.
- Range of operation is limited.

2.2.3. Future Prospect

By using the original code, the robot could extend its range and do a more complicated path thus having the possibility to get all the boxes in the maze.

Also by doing that it could always have 3 boxes in the hold before getting to the stacking area without having to start from a specific corner.

This robot could very well be made into an industrial stacker, but the original strategy should be used. Also more sensors should be added in order to get more feedback from the environment and simplify the code.

Appendix I
BOReS: the Omitted essential systems

The BOReS is divided in 2 segments. One part is intended to avoid collision with an obstacle when freely navigating away from a wall while the other to recognise boxes that aren't in front of the robot.

ORes (**O**bstacle **Re**cognition **S**ystem)

The ORES code was to be implemented as soon as the searching pattern was broader than wall following. In order to scan the center area, this code would able been crucial for avoiding an obstacle in front and maintaining the Robot's path until it came back across. It also takes into account that a wall is not an obstacle.

(This code is 90%~80% completed and 80% tested)

Inputs: odometer and sensors readings

It has several versions. In older versions, the ORES is a task and in newer is a function called form the main task in an event of an obstacle or wall:

- Obstacle is in front
- Memorise path heading
- Check odometer for position
- If you are to close to X min, X max, Y min or Y max and the heading is -90, 90, 180 and 0 respectively it means you are seeing the wall and not an obstacle and should not try to continue its forward path, but change the heading by rotating to 180, 0, 90 , -90 respectively.
- If the previous condition if false, turn left until you see no more obstacles then turn and extra 45 degrees to the left to let the whole body of the robot pass
- Drive forward the max length of an obstacle or a bit less than what the HUS is showing if that value is less then the max length of an obstacle
- Turn right and start reading the HUS until you read a small value (i.e. search the obstacle you just passed)
- Turn left 45 degrees from the current heading and drive forward the max length of and obstacle or a bit less than what the HUS is showing if that value is less then the max length of an obstacle
- Turn to the path heading you memorised at the beginning and exit function

A later version should have used the infrared sensors as well for wall detection and the function would have returned to a task taking over the control of the robot if an obstacle was too close to prevent hitting the obstacle /wall and adding error to the odometer.

BOReS (**B**ox and **O**bstacles **Re**cognition **S**ystem)

The initial purpose of the BOReS system was to identify boxes and obstacles, avoid the obstacles and pickup the boxes. The purpose of BOReS I was limited to identifying "pick-able" boxes within 80 cm to the left of the robot, determining their relative position, and loading them onto the robot, after which returning the robot back in the position from which is detected the box.

To solve this problem a rather sophisticated approach was required. Due to the poor reliability of the ultrasound sensors, a carefully designed and thoroughly tested algorithm needed to be implemented to compensate for their flaws. The main problem with the sensor was that it would see the box much before it was in front and this varied with the orientation of the box (side, or edge towards the robot's path). As far as hardware, the low us-sensor needed to be placed looking on the left side of the robot (5 cm height), while the high us-sensor needed to be placed looking in front (and at about 20 cm height).

The main idea of the algorithm is to monitor the low us-sensor and stop the robot once it detects an object with the range of about 80 cm. The robot will then rotate counterclockwise monitoring the high us-sensor. If the high us-sensor reports anything within the distance of the spotted object + 20cm, then the robot will skip and restore orientation. If the high sensor does not report such an object, then the robot will rotate back (clockwise) while monitoring the low sensor to find the first and last occurrence of the box, and record the angles of these two occurrences. The average angle will represent the angle at which the box is located. The robot will turn to that angle, and drive close to the box. At this point the robot will use the infra-red sensors to get the box at close range. After the box is successfully loaded, the robot will compute its relative vector position back to the position where it saw the box, return there, and re-orient. From this point the control is restored to the main program, that would resume driving and restart the BOReS system.

While this system proved very reliable in collecting boxes within 80 cm to the left of the robot it was never integrated to the final product, so it never reached a final version.

There are two main versions of code:

BOReS I - v0.5 – fully tested and bug free, but stand-alone, only proves the reliability of the implemented algorithm, provided there are no obstacles in **front** of the robot.

BOReS I - v0.6 – this version is ready to be integrated with a system that would provide a trajectory, and take care of any obstacles and boxes in the front of the robot (i.e. navigation and obstacle avoidance, these also never reached the final product)

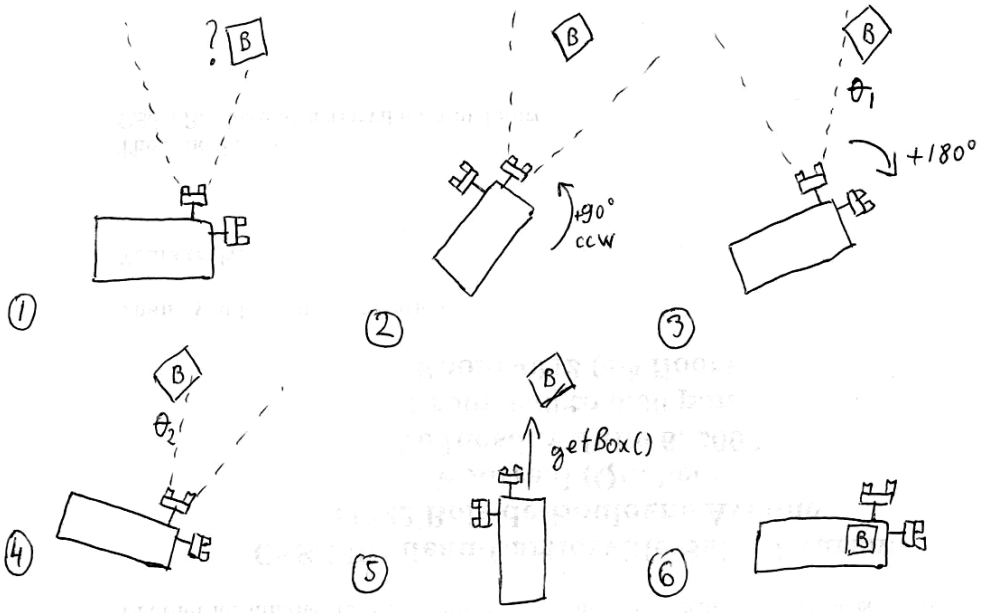
This code uses some auxiliary code found in other files, the description of that code is present in their files, but briefly they are as follows:

- Functions to drive forward, to rotate at an angle, to return to a position, to access the sensor readings (for the i2c_control), to approach, and load the box at close range.

- Odometer, that runs as a task and provides reference of angles and positions, it should normally be corrected by other functions of navigations (all never integrated in the final product).

The code is about 150 lines without the auxiliary code and without comments.

See the diagrams below for the illustration of a situation when a box would be identified and collected.



1-Unknown object seen by the low US sensor on the left of the robot
2-Look with the higher US sensor to make sure it's not a wall
3-Record the angle of the left edge of the box
4-Record the angle of the right edge of the box
5-Compute center angle and move forward. If box is not picked up 15cm passed US recorded distance, jiggle robot and move forward another 20cm
6-Return to initial position and resume sweep pattern

Appendix
Time use over project life

To following table is a summary of time used. Displayed in color is the task, and the red bar to its right is the actual hours used. The bright green bars represent the cumulative over/under hours.

Usage	Week 1	Week 2	Week 3	Week 4	Week 5
Structure	18	18	18	15	0
Used	10	15	6	11	0
Navigation	20	20	20	20	0
Used	8	6	25	26	
Recognition	20	20	20	20	0
Used	7	7	23	28	0
Integration	0	0	0	0	15
Used	0	0	0	0	38
Debugging	0	0	0	0	15
Used	0	0	0	0	47
Paper work	2	2	2	5	0
Used	2	2	7	5	0
Over/Under	34	30	-1	-10	-55
YTD	34	64	63	53	-2