**SYSTEM DOCUMENT**

Project: DPM Robot Design and Competition

Task:

**Document Version Number: 7**

**Date: April 6, 2015**

Author: Team 18

Note: This week’s changes are made in magenta. Previous changes are now in black.

1. **TABLE OF CONTENTS**

* 2.0 System Model
* 3.0 Hardware available and capabilities
* 4.0 Software available and capabilities
* 5.0 Compatibility
* 6.0 Reusability
* 7.0 Structures

7.1 Mechanical Structure

7.2 Software Structure

* 8.0 Methodologies
* 9.0 Tools
* 10.0 Testing
* 11.0 Glossary of terms

**2.0 SYSTEM MODEL**

(*Once you have fully understood the problem through the Requirements, you can have a first pass at trying to figure out what the basic components are that will be needed. These can be sketched out in a block diagram which is the first step in a Functional diagram of the system –just a sketched out block diagram of the basic functions you think might be needed. It should gradually be updated as the implications of the client requirements become clearer. This model is needed to try to understand how the requirements and solution can be mapped onto the environment you have been given to solve the problem. This may also depend on the human resources- team capabilities – you have available*)

* A flowchart diagram has been added with the appropriate linkages.
* Essentially, we will require the robot to be designed and built, which the software depends on. Then we will enter testing phases, and finally the integration phase. Between these there is likely to much tweaking and updating of documentation.

Hardware and Software Units (listed)

Software Architecture

Although, these flowcharts represent the initial plans of the team, eventually the team never used these strategies for the robot. Please see the section 7.2, Software Structures for more details on the strategy used on competition day.

**3.0 HARDWARE AVAILABLE AND CAPABILITIES**

(*In the sense of the design problem you have, there are three hardware definitions. First, the Lego components and their mechanical capabilities. The sets limit the structures you can build. What are these limitations? Second, there are the electromechanical limitations. Third there are the electronic/processor constraints (e.g. – how fast can you execute code?). All these constraints need to be identified and listed*.)

-Constraints document mentions some limitations

-The batteries used in the NXT will have limitied voltage, most likely 1.5V, so this sets a limit on the speed of the motor rotations.

**i) NXT Brick\***

* 32-bit ARM7 microcontroller @ 48MHz (256 KB FLASH, 64 KB RAM)
* 8-bit AVR microcontroller (4 KB FLASH, 512 Byte RAM)
* Bluetooth wireless communication (Bluetooth Class II V2.0 compliant)
* USB 1.1 full speed port (12 Mb/s)
* 4 input ports, 6-wire cable digital platform
* 3 output ports, 6-wire cable digital platform
* 100 x 64 pixel LCD graphical display
* Loudspeaker (8 kHz sound quality). Sound channel with 8-bit resolution and 2-16 KHz sample rate.

One brick can use up to 3 (three) motors.

**ii) Power source**: 6 AA batteries

**iii) Light Sensor\***: Measures a normalized light value or in percentage form.

**iv) Ultrasonic Sensor\*:** Measures distances between 0 and 255 cm, +/- 3 cm.

**v) NXT Motors:** Limitations in terms of maximum torque and speed prevents the use of certain launching systems such as the 'fly wheeler'.

**vi) Wheels:** Limitations include:

* Tire slipping
* Actual radius cannot be measure directly but need trial and error method to adjust its value
* The small tireless wheel often does not turn (especially at corners) adding to friction.

(\*Source: [*http://nxt.cs.uwindsor.ca//499football/features\_limitations.pdf*](http://nxt.cs.uwindsor.ca//499football/features_limitations.pdf))

**4.0 SOFTWARE AVAILABLE AND CAPABILITIES**

(*The software systems you can construct are constrained by the tools and languages you have access to. What Tools are there? What are the advantages and disadvantages of each one – e.g. ease of use versus speed of code execution, generality of operation versus size of code, etc.*.)

**i) Java**

- adv: Most team members are capable of the use of basic java.

- disadv: Slower than other lower level languages; uses too much memory.

**ii) Eclipse IDE**

- adv: Very versatile.

- disadv: Debugging is difficult, especially when working with Lejos. Slow booting.

**iii) NXT Lejos**

- adv: Allows familiar programming platform (Java).

- disadv: Poor API documentation, not enough information.

The software systems are likely to be constrained and bound by physical realities of real world environment and must therefore take into such factors. Trial and error and plenty of testing may need to be done.

**5.0 COMPATIBILITY**

(*Adherence to requirements within the design environment, e.g. Lego components plug together in certain ways, so everything has to adhere to this. What about software? Are there any compatibility issues there? Will you connect to third party systems, etc.? In this area, you might want to list pieces of code or mechanical structures that have been developed in the lab – these speed up development time but might place constraints on how the system will function, i.e. they were constructed with certain assumptions – you need to know what those were so that you can make sure that they don’t conflict with the current intended usage.)*

**Hardware:**

* Pieces from different kits/groups were taken and built into a new design structure so compatibility between the previous reusable software systems and the new hardware system is an issue e.g. radii of wheels is affected by the structure of the entire design and is only measured through trial & error. The radii values used in previous navigation & odometer codes are no longer valid, since the structure and weight of the design has changed. These have to be adjusted in the codes.
* Compatibility between the previous launcher system and the new reloading system has to be addressed.
* Materials not provided in the kit such as rubber bands, tapes or strings may be used. These need to be compatible with the rest of the Lego pieces.

**Software:**

* Between code algorithms of different laboratories e.g. polarity of clockwise/anticlockwise rotations, angle measured in theta and radians, etc.
* Measurements of hardware pieces such as:

- distance between wheels,

- distance between sensors and robot centre,

- wheel radii

used in codes need to be adjusted and made compatible to the new structure.

**6.0 REUSABILITY**

(*Structures which may be useful for several parts of the system, existing mechanisms and sub-assemblies which might be used, existing software, etc. The labs might produce some of this – if so, what? – relates to the previous section as well.)*

**Mechanical Systems:**

Flicker Launcher system from Lab5.

**Software Systems:**

Wall Follower from Lab1, Odometer from lab2, Navigator from lab3, Localizer from lab4.

**7.0 STRUCTURES**

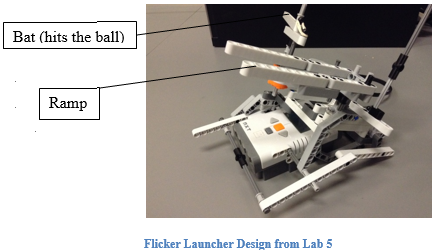
(*Mechanical structures – reasons for choices, etc., electrical structures, design of software structures, etc. Some of this will derive from the constraints sections earlier. The rationale for choices must be given – this will allow a critical review of decisions, etc., before the design progresses too far or real physical systems are built.)*

**7.1 MECHANICAL STRUCTURE:**

*In this section: 1) Core Structure (Flicker Launcher & Patbot + Reloading Mechanisms), 2) Modifications.*

1. **CORE STRUCTURE:**

Flicker launcher system (from lab5). Planning to use only 1 NXT brick for launching and navigation carried by two large wheels and two small black wheels that are used for balance. Reloading mechanism is being developed (please see the next section for that).



*Why use 1 NXT brick?*

Light weight design, enhanced motion, easier to maintain all the functions of the robot. Avoid the complexity of multiple NXT bricks.

*Why use two large wheels and not more?*

The 4 wheel idea was considered but it is difficult to operate such a system with turns. Two wheels are easier to use in this regard.

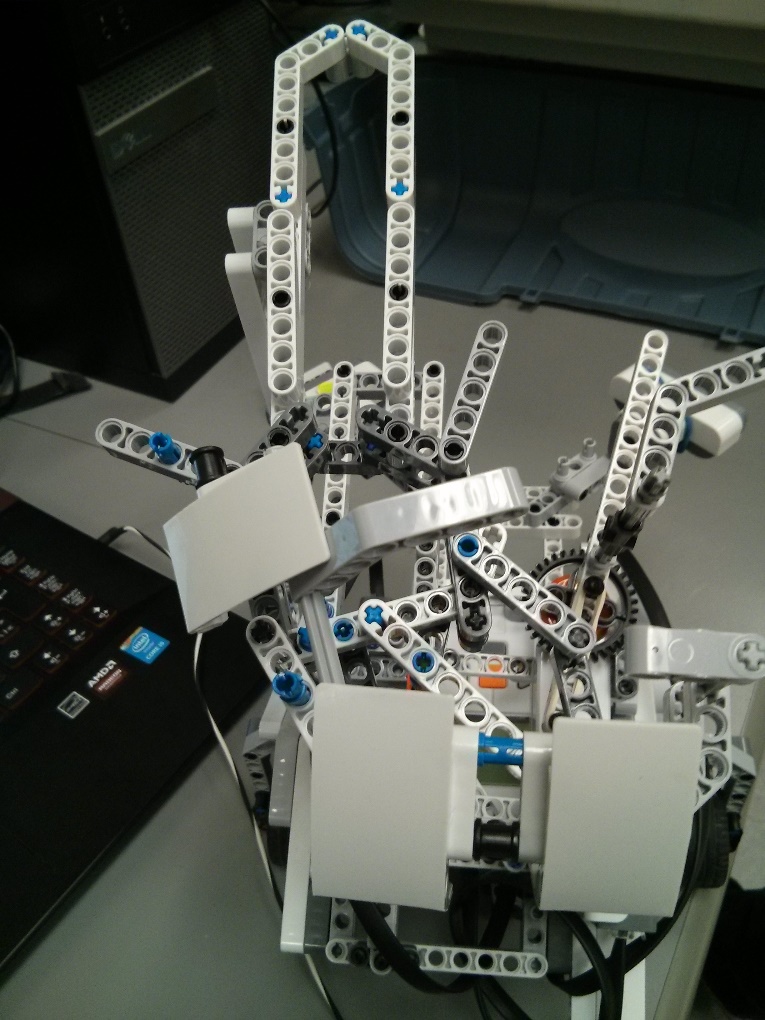
*Why the flicker launcher system?*

\*For more details on how they work and the pros and cons of the three main launcher designs, please refer to the document ‘Launcher Designs Analyzed’ in the ‘Mechanical Designs’ folder.

*How many of what type of sensors are being used and why?*

The mechanical team tried many different combinations of sensors over the past weeks on the robot for the best result with regards to the teams design and strategy in accomplishing the objectives. Please see the Modifications section below to see what changes were made and why.

**2) RELOADING MECHANISM:**

**The robot should be able to shoot and reload on its own (see Requirements document for details on this) once it starts to move. While the launcher and the motion systems were taken from previous lab works, the reloading mechanism was a new concept that needed development. The design team came up with two different reloading mechanisms, before bringing the two into one final design, in an effort to combine the pros and eliminate the cons of the two separate designs.

Reloading design 1

**i) Reloading Design I (JOSH):**

*Idea*: place balls on an inward curve, and as the motor rotates, a piece pushes them along and into the loader. If the loader is already full, the excess balls go onto the overload ramp.

Reloading Design I: Reloading mechanism on top of flicker and two wheels at the bottom.

*Issues faced:*

-Shaking and instability of the design structure.

-The stacked balls displace the one at the bottom just before it is fired such that the ball is not launched properly. The weight and angling of the additional balls may cause the bottom ball to move forward a little; this leads to inconsistency and inaccuracy of launching trajectory.

-It is much more practical to create a launcher that uses gravity to load the balls into the launching mechanism. This launcher pushes the balls over a cavity and then allows for overlap; this is not productive and is addressed in future designs.

**ii) Reloading Design II (MATHIEU):**

Idea:

* Implement a reloader with a “stopper” that will prevent balls from piling up on top of the one being shot. This causes faulty launches and could lead to error.
* The idea for this design was to have something constantly blocking the drops, let a piece attached to the rotating axel “free” the path to let one ball through. Horizontal line up of the balls at an angle.
* The stopper would then return to its initial position with the help of an elastic.

Issues:

* Instability lead to incapability of successfully moving the blocker arm, sometimes balls wouldn’t fall through
* Hard to stabilize since most of the area needs to be kept free so other pieces can rotate across.
* Ensuring arm pushes enough so that gap is big enough to let balls through.
* Ensuring sure balls fall in the correct place
* Ensuring balls do not fall off their holder (top part)
* Keeping the whole structure centered to avoid tipping

Reloading Design II

**iii) Reloading Design III (COMBINATION):**

Idea:

* A continuation of previous designs
* Keep the “stopper” concept, but place it as a circular disk on the rotating axel of the flicker.
* Instead of having a piece push something to free space for the balls to leak through, a gap in the blocker was made such that it will let one ball pass per rotation.
* Attempted using a CD, cutting out a slice.

Problems:

* “Jamming” causes motors to stop if misplaced ball blocks the rotation (Major issue)
* Getting piece to fit between the flicker and reloader without impeding the motion of the launcher or getting stuck on stationary pieces.
* (Previous): Stability, consistency (no jams), accuracy (1 ball, not 2), fall area.

**3) WEEKLY MODIFICATIONS (made on robot):**

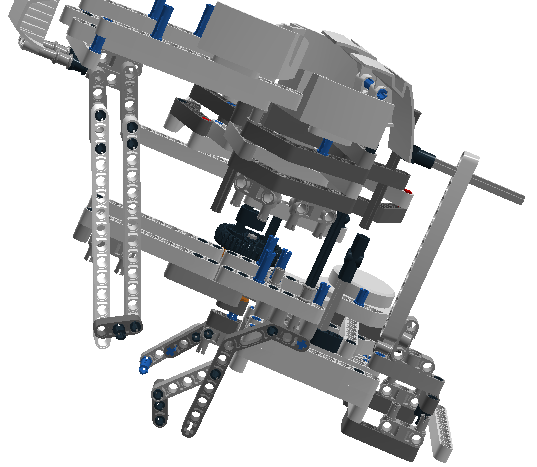
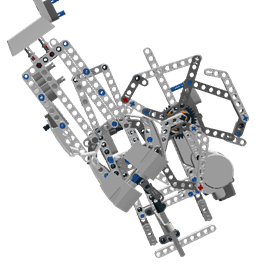
**Update (week 4, March 16): Flicker Launcher** (now on robot)**:**

For compatibility with the added reloading mechanism, the flicker system has also been modified with small changes:

-Shortened ramp for improved overall stability of the structure.

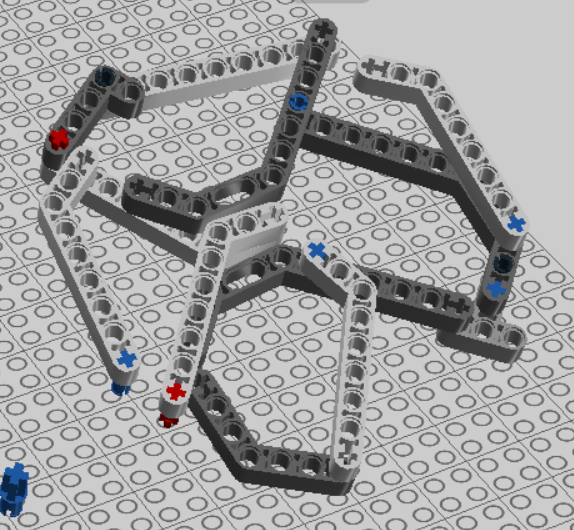
-Pieces added to hold in place the ball that is to be fired.

-A disk like structure added to prevent the other ping pongs from weighing down on the one being launched. Makes sure only one ball passes at a time to be launched. Diagrams are provided below:



Ramp under development (view from above)

Ramp under development (view from side)

****

disc like structure

**Update (week 4, March 16)**: Now we have 3 Ultrasonic sensors at the front. The idea is to remove any chances of hitting an obstacle by not having any room for 'blind spots' of the ultrasonic sensor.

**Issue**: Will the 3 sensors work harmoniously without conflict?

**Update (week 6, March 31):** Some pieces were added or changed for stability. In particular, a piece was added on the loader to prevent balls getting stuck. For more details, refer to X.

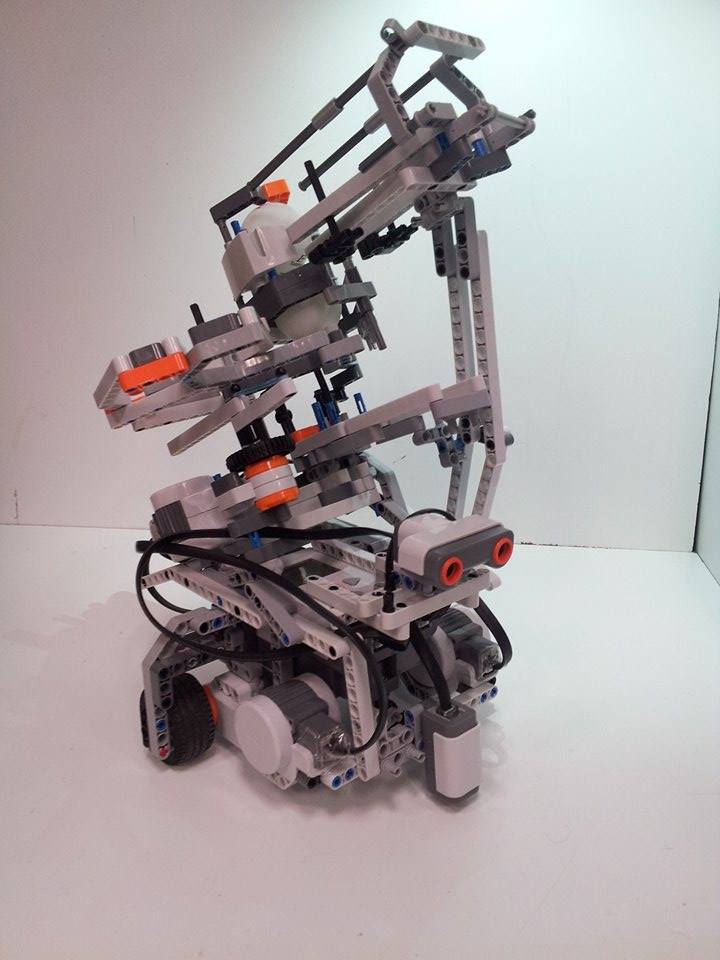
**Update (week 7, April 6)**: 1 Ultrasonic and 1 Light Sensor at the front. Their purpose will only be to localize the robot. Ultrasonic sensor will not be used for obstacle avoidance and the reason for this is explained below:

The use of ultrasonic sensors was initially considered to be a means of avoiding obstacles and reaching the destination. However, the project requirements do not make it mandatory to use ultrasonic sensors as the only means for obstacle avoidance. From the requirements document we know that:

*“To “pass”, a robot must successfully demonstrate each of the capabilities (localization, navigation, obstacle avoidance, reaching the shooting zone, firing a single ping-pong ball at one target, returning to (0,0)) at least once during the 3 runs. The winner of the competition is the team that has the most points at the end of the day.”*

The map consists of obstacles whose positions are known and those whose positions are not known. After weeks of testing, the team decided that it would be easier and more efficient in terms of design, coding and accomplishing the overall competition objectives to use a different strategy for obstacle avoidance, one that would not involve using ultrasonic sensors.

This strategy is to simply take a predefined path to reach the robots destination. Please look at the ‘Software Structures’ section (below) for more details on this strategy:



Final Design (as of April 6th)

**7.2 SOFTWARE STRUCTURE:**

*This section consists of:* ***i) Strategy, ii) Software Log & iii) Class Hierarchy.***

1. **Strategy:**

*Basic objective outlined below. Refer to the ‘Requirements’ file in () for more details.*

*Task: “An autonomous robot will be designed, tested, and created using a Lego NXT kit. It will need to be programmed, with the LeJos language, to localize its position and navigate on a field with obstacles, and then to launch ballistics (ping-pong balls) at a defined target outside the grid. The robot must be capable of identifying its position and navigating to specific points from other points within a square grid. The obstacle course is to be completed in the shortest time possible, with the robot being involved in a competition with a defined set of rules.”*

Furthermore:

*“To “pass”, a robot must successfully demonstrate each of the capabilities (localization, navigation, obstacle avoidance, reaching the shooting zone, firing a single ping-pong ball at one target, returning to (0,0)) at least once during the 3 runs. The winner of the competition is the team that has the most points at the end of the day.”*

We want the designed robot to localize itself at its initial position, head towards the location from which it can launch ping pongs, launch ping pongs and then return to the starting position.

* **Localization: i)** Ultrasonic Localization of type Rising\_Edge to get an approximation of its location with respect to the grid lines. This localization is less accurate due to the sensor being of a low order resolution.

**ii)** Data obtained from US Localization is used to perform the light localization. Light Localization has higher accuracy because of the light sensor resolution being of higher order.

* **Navigation:** A method called travelTo() makes use of a loop to constantly check whether the robot has arrived at the desired destination or not. This strategy has:

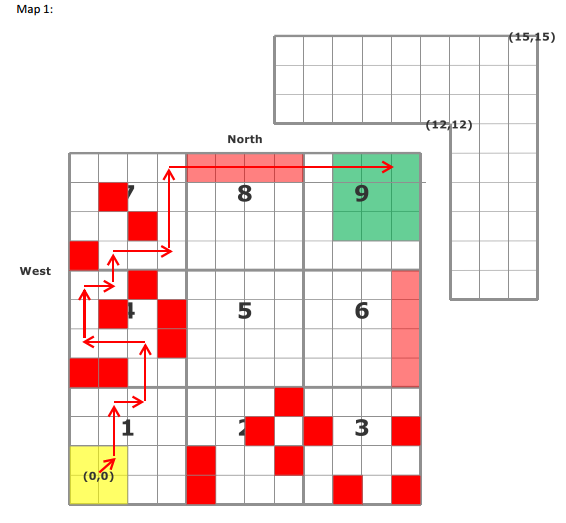
Pros: -can interrupt travel, useful for obstacle avoidance.

-can immediately make use of the corrections made by odometry correction to constantly correct its heading and position.

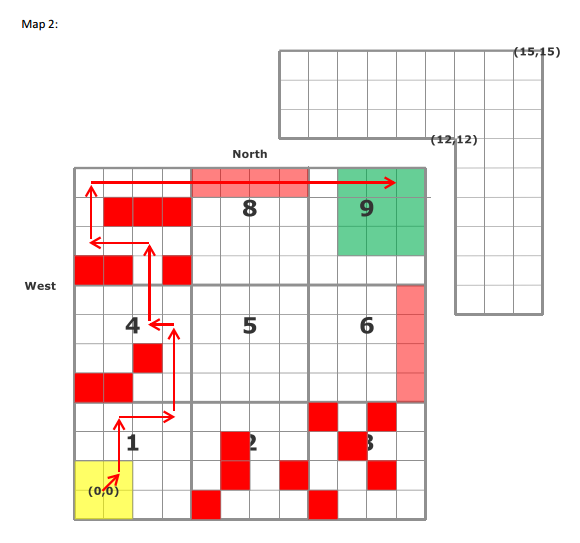
Cons: -inherent error, due to an acceptable threshold of arrival.

The robot will NOT be using ultrasonic sensors for obstacle avoidance. The strategy to avoid obstacles and arrive at the destination is by taking a predefined path through the known positions of obstacles. The design team finds this easier to implement than using ultrasonic sensors. This strategy would help accomplish the goals of the competition with more certainty than the strategy of using ultrasonic sensors for obstacle avoidance.

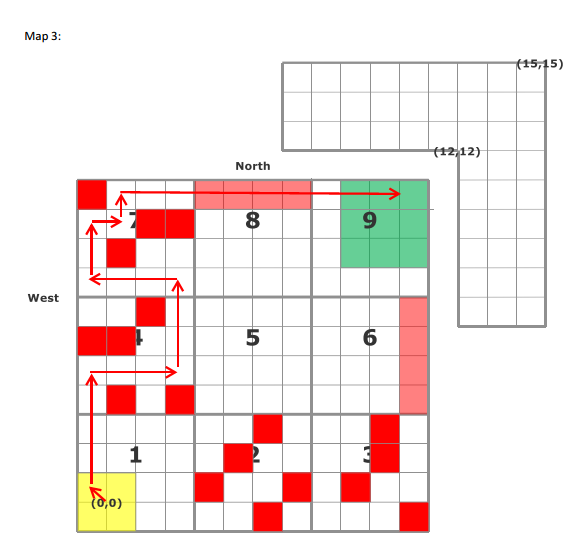
According to the updates in the project description, (see Requirements doc of this week), robots will be given 1 out of 3 maps to navigate and random target coordinates to shoot at for each of the 3 rounds. As explained above our team plans to take predefine paths to make the robot arrive at the location from which it can shoot. The path ways for each map are provided below. The arrow heads are the waypoints of the total path.



Pathway for Map 1



Pathway for Map 2



Pathway for Map 3

* **Launcher:**

A function is used to control motor speed on call, consequently making the launcher perform the launching sequence.

A Method has been created that takes in as input 2 integers (x,y) coordinates of the target, which is then used to calculate the position set of the robot (x,y, and angle at which the robot needs to be) in order to perform the launch. NOTE: these integers are in the grid format used in the documents e.g. an input (10,10) is actually referring to the grid intersection (10,10) which in cm would be (300,300) on the playing field.

*(Please refer to the section ‘System Model’ above for a visual of the software components and a flowchart of the software architecture)*

**Weekly Updates:**

**Issues (as of 1st April)**: Primarily with odometer correction. Odometer calibration needs to be very accurate and this seems to posing some difficulty. If odometer correction begins to work correctly, the robot navigation would be accurate and the robot would be able to reach its destination even if it has to travel long distances.

**(As of 6th April):** Odometer calibration done to sufficient degree of perfection to enable accurate enough long distance navigation.

1. **Software Log:**

- Implement Navigation (Victor/Musa’s code) on the robot, which worked sufficiently.

- Implement US&Light Localizer, struggled because the odometry used for navigation does not work for localization & vice versa. Spent some time to make either one to work on both.

- Decided to change Navigation code to the one Bobak/Clara wrote, because the code appeared to be compatible with both Localizer and Odomtery.

- Corrected equations in Localization code to compensate for errors in the y-axis.

- Began testing navigation for the given Beta Demo Floor. Found too large errors and then task the testing group to determine accurately the radii of the wheels.

- Simultaneously testing was performed on the launcher components in order to determine the launchers shooting distance, offset angle and accuracy.

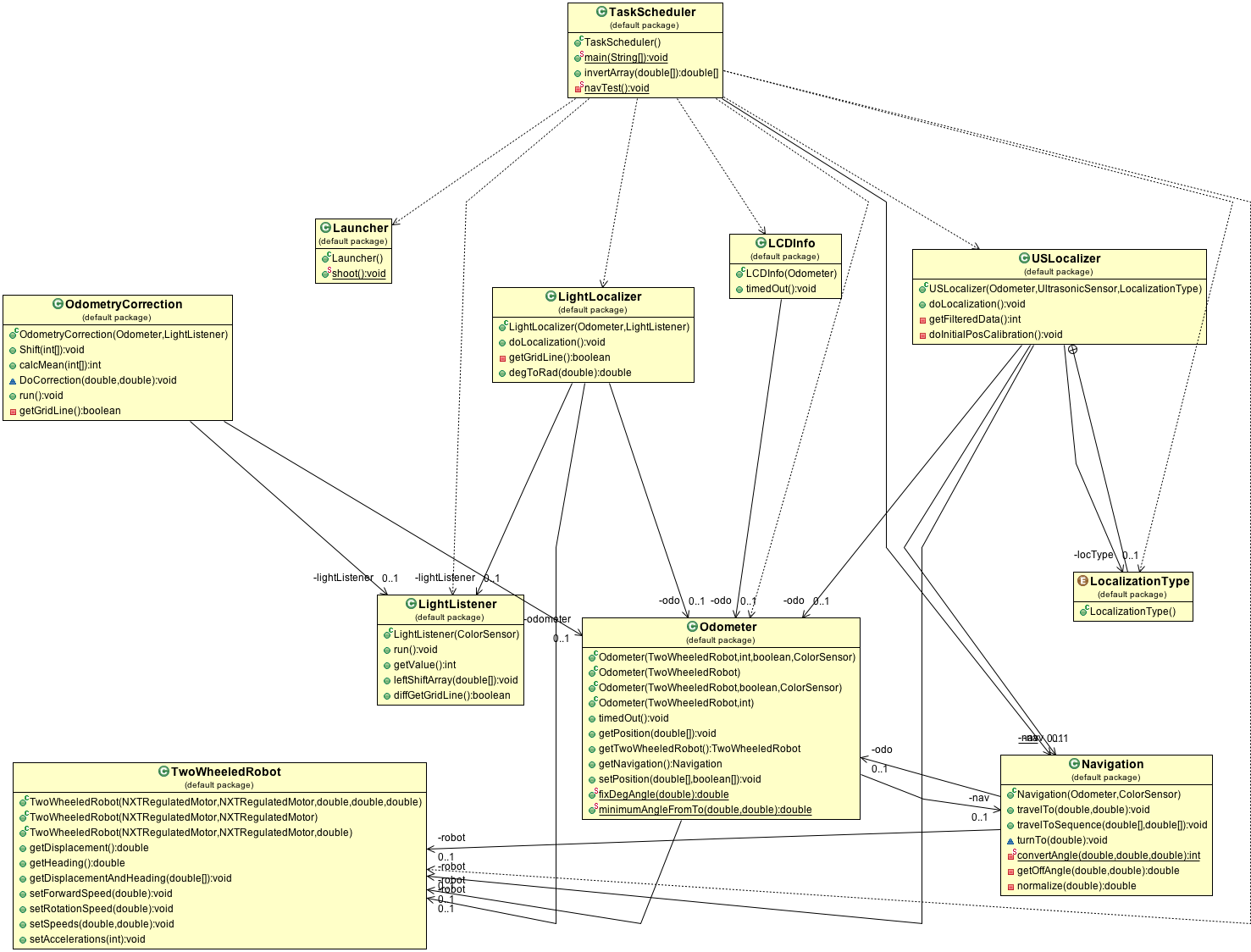
- (For Beta Demo) while waiting for testing results to be conclusive, we have temporarily scaled hardcoded values to compensate for errors in Navigation.

- Implemented the tested results for the odometery values (wheel radius and wheelbase) Results were tweaked to achieve greatest possible precision of odometer.

- Implemented the new values for lightsensor localization (color sensor distance from odometery center). Tweaked to achieve greatest possible precision after second phase of the localization is complete.

1. **Class Hierarchy:**

* **~~DPM\_FINAL:~~** ~~Acted as a task scheduler, creating Odometry, Display & Navigation objects~~
* **~~Navigator:~~** ~~Used two methods, turnTo() and travelTo() in order to navigate to coordinates given in DPM Final.~~
* **~~Odometer:~~** ~~Provided odomtery based on wheel rotation.~~
* **~~Odometry Display:~~** ~~Allowed display of info (such as coordinates) on the NXT screen.~~
* **Task Scheduler:** Replaced DPM\_FINAL from last week; similar function.
* **Two Wheeled Robot:** Object which holds all robot specific constants and values i.e. wheel radii, wheel base, speed, acceleration, width.
* **US Localizer:** Performed basic localization using Rising Edge.
* **Light Localizer:** Performs more accurate localization using light sensor.
* **Launcher:** Contains a public shoot() method.
* **LCD Info:** Displays info onto LCD screen.
* **Light Listener:** Polls light sensor for data.
* **~~Light Sense Listener:~~** ~~Same as listener, but with data logging capabilities for testing.~~
* **Navigation:** Bobak and Clara’s Navigation system, uses two methods, turnTo() and travelTo().
* **Odometry:** Odometry reliant on two wheeled robot (Bobak & Clara)
* **Odometry Correction:** Correction using light sensor
* **~~US Listener:~~** ~~Polls US sensor for data.~~
* **Localization Type**

****

UML Diagram of the Software Structure (please refer to the folder: ‘Software’ and open the image file UML Diagram for a clearer version of this.

**8.0 METHODOLOGIES**

(*Approaches being taken in all parts of the design and, for software, the basic algorithms to be used. These are really lists of the possible candidate solutions for parts of the problem. They come out of the Ideas Generation phase and will allow a critical analysis before the final design is performed*.)

* Break down the entire task into interdependent sub tasks. Reuse codes and mechanical structures of previous labs for each of these subtasks. Combine everything into a single mechanical structure capable of performing the various tasks necessary to complete the required goal.
* Test each of the main components that we are making use of and which have a significant impact on the robots functionality e.g. tires, light sensors, ultrasonic sensors, rubber bands, motors, nxt brick, etc before they are assembled. The idea is that if individual components are performing according to plan, the final product should also perform well.
* Documentation process progression for mechanical design:
  + Images – Initial idea, good to get a view of the structure.
  + Videos – Pictures were very restricted, didn’t get a full view, could not reproduce.
  + Step-by-step build (pictures) – Taking pictures of the building process in a step-by-step manner allowed us and others the opportunity to reproduce the product. Still very restricted view.
  + LDD – Having a 3d model of the final design allows users to get a 360 interactive view and observe from any angle they want. Furthermore, the building instructions are premade in the program. However, LDD only accepts perfect connections which caused a lot of problems for our model. To overcome this we took pictures of any discrepancies and explained in the “misconnections” folder found in the same location as any LDD representation of our models.

**9.0 TOOLS**

(*Details of the tools available to be used to construct the system and why might be useful. Again advantages and disadvantages of the various tools might be given. This might relate back to section 3. If so, do not repeat it here – use a pointer to the information given previously…)*

Main tools are:

**Hardware:** NXT kit, rubber bands, strings, ping pongs, tapes.

**Software:** Lejos API Library, Java API Library, Microsoft, Lego Digital Designer (for visualizing design, editing & planning).

Issues: Compatibility issues between tools external to NXT kit and kit itself. Difficult to control elasticity of rubber bands.

-Will be using Gantt chart to organize schedule in coherent manner

-Currently using Facebook and email to communicate

-Will submit documents to professors using Dropbox.

-Dropbox is being used to store and edit all documents, so that all members have access to the latest version.

- Github is to be used to manage and share the software.

**10.0 TESTING**

We will first test the robot in the DPM lab using test cases in Java. We will record the results using word and excel. A document which outlines our testing plan can be found in the folder named ‘Testing Plan’.

Various tests done this week have been mentioned below. The details of each test are given in detail in the ‘Testing Document’ and can be found in the folder named ‘Testing’.

**Hardware Testing:**

1. Date: 26th February

Hardware version: Launcher1 (Flicker1) (Mathieu/Josh)

Goal: Determine the distance, trajectory angle, and accuracy of the ball launcher. Note: not yet on robot.

1. Date: 27th February

Hardware version: Launcher 1 (Flicker 1.1)

Goal: Determine the distance, trajectory angle, and accuracy of the ball launcher.

1. Date: 27th February

Hardware version: Launcher2 (catapult 1.1) (Clara/Bobak)

Goal: Determine the distance, trajectory angle, and accuracy of the ball launcher.

1. Date: 11th March

Hardware version: (N/A) Ultrasonic sensor

Goal: Determine the accuracy of the distance detected by the ultrasonic sensor for straight lines and angles.

1. Date: 19th March

Hardware version: Yo-o 1.1

Goal: Determine the distance and accuracy of the ball launcher.

1. Date: 19th March

Hardware version: Yo-o 1.1

Goal: Determine the distance and accuracy of the ball launcher

1. Date: March 19th, 2015

Hardware version: Yo-o 1.1

Goal: Determine if the three ultrasonic sensors interfere with each other.

1. Date: March 25th, 2015

Hardware version: Yo-o 1.1

Software version: Soft V0.3\_r2

Goal: This test will be used to determine the landing coordinates of our launcher. We will use these values to determine whether our launcher is accurate enough. We will also use the obtained results to determine where our robot needs to launch from.

1. Date: March 26th, 2015

Hardware version: Yo-o 1.1

Software version: Soft V0.3\_r2

Goal: This test will be used to determine whether our odometer readings are accurate or not. Since the navigation is reliant on the odometer, an accurate odometer will yield successful navigation.

1. Date: April 2nd, 2015  
   Hardware version: Yo//o 1.2  
   Software version: Soft V0.3\_r2  
   Goal: Previously we ensured the radii proportions were correct by verifying that the robot would travel in a relatively straight line. The next step is to correct their absolute values to make sure the robot knows how far it is travelling.
2. Date: April 2nd, 2015  
   Hardware version: Yo//o 1.2  
   Software version: Soft V0.3\_r2  
   Goal: This test will be used to determine whether our wheel base value is accurate or not. Having a proper wheel base value will make our odometer more accurate on turns and allow our navigation to work properly.
3. Date: April 3rd, 2015  
   Hardware version: Yo//o 1.3  
   Software version: Soft V0.8  
   Goal: This test will be used to determine whether our light sensor distance is accurate by making the robot run over a line and telling it to stop when the center has reached said line.
4. Date: April 9th, 2015  
   Hardware version: Yo//o 1.3  
   Software version: Soft V0.8  
   Goal: This test will be used to determine whether our light sensor distance is accurate using our localization. This will be determined by observing whether the robot travels to the proper (0,0) coordinates after its localization.

**11.0 GLOSSARY OF TERMS**

(*Again, define all the terms you are using so that all members of the Team and any outside readers understand what you are talking about*…)

**API –** application programming interface

**brick –** LEGO Mindstorms NXT microcomputer unit

**port –** input/output slots on **brick**

**launcher downtime-**

amount of time that the launcher needs between shooting cycles, not taking into account the time needed to reload.