



Challenge Design Document

Team 6: Thomas and Friends

INTRODUCTION

Problem Statement

The 2015 RSS Grand Challenge is to build and program a robot that will autonomously construct a shelter given several colored blocks placed in a maze of obstacles. The robot will be given a prior map of the obstacles. The robot's task is to find and collect blocks for its construction task and then transport them from their start locations to an autonomously determined construction location where the robot will build a shelter. The shelter can take any one of several forms, including a low wall, a multi-level wall, an "L" or "V" shape or a room-like configuration.

Overview of Approach

We plan on approaching the RSS Grand Challenge by designing and programming a robot that can build 3 x 2 standing towers of blocks. We plan to use the robotic arm to pick up blocks and insert them into a funnel at the back of the robot which will deposit the blocks into a storage unit. The storage unit will align the collected blocks so that, once six blocks have been retrieved, they form a 3 x 2 tower of blocks. Once the blocks have been collected, the robot will navigate to a construction site, where it will release the 3 x 2 tower from the storage unit. After releasing the tower, the robot will continue to collect blocks and deposit them in the storage unit while wall-following.



Assumptions

Our assumptions in designing our approach to the RSS Grand Challenge are that:

1. The terrain is a flat surface with walls separating parts of the challenge region
2. Walls delineate the borders of the challenge region and that all walls are tall enough to be recognized by sensors placed several inches above the ground on a robot

3. The prior map given provides the positions of a subset of the internal walls physically present as well as exact positions of the borders of the challenge region
4. The prior map of the challenge region provided does not necessarily give any precise information about the location of blocks
5. The blocks are sufficiently different in color from the walls and floor
6. The amount of time given to complete the challenge is on the order of several minutes



Team Focus

The primary focus of our team is to produce a challenge implementation that works robustly and can complete a defined task that falls within the scope of the challenge. We would rather have a simple implementation working effectively than have a complex implementation working unreliably or not fully working.

To make sure we fulfill this overall goal, we have outlined two options for our software implementation in the technical approach section below, as well as an alternative mechanical design to fall back on in case our primary design fails. The first software implementation plan is our desired implementation, while the second is an alternative simpler implementation that we will change to if we find it infeasible to meet the milestones for our primary implementation.

Aside from producing a working system for the challenge, we also would like to use the Xtion sensor to implement some robust form of localization. As indicated in the technical approach section of this Challenge Design Document, if we end up getting the primary implementation working, then we will use the Xtion-based localization for motion planning during exploration. Otherwise, if we go with the second implementation, we will use the Xtion-based localization to implement a map-informed wall following method.

TECHNICAL APPROACH


Overview: Mechanical and Electrical (Erin and Ryan)

We have devised two potential implementations of the mechanical side of our approach to the challenge. Most of this document is written assuming that we will successfully use the primary implementation plan. More information about the alternative structure can be found towards the end of the document in the “Alternate Mechanical Design” section.

1. **Primary Design:** Our primary design for block collection includes a hopper to collect blocks in a 3 x 2 tower behind the robot, a funnel feeding blocks into the hopper from on top of the robot, and actuation of the robot arm used previously in course labs to deposit blocks into the collection system.

2. **Alternate Design:** If our hopper mechanisms fail to work reliably as we approach the Grand Challenge date, we will replace the design with a simple structure which funnels blocks into a straight line underneath the robot base as the robot drives over them during the challenge.

The mechanical design of our robot will consist of sensors to read the environment, drive motors to propel robot movement, a camera feed for identifying blocks, and actuators for the collection, storing, and arrangement of the blocks. The key components of the mechanical design are described below.

1. **Environmental Sensors:** The robot is equipped with four sonar sensors, two on the left side and two in the front. These sensors read the distances to obstacles that will be used in wall-following. It will also have bump sensors on board to prevent running over objects.
2. **Drive Motors:** The robot has two GM9236S025 DC motors attached to the front wheels in order to control movement. The motors receive a PWM voltage waveform for input from the uORCBoard and move accordingly. Each motor also has an attached quadrature encoder to measure the distance driven. For stability, the back of the robot is equipped with drag wheels. 
3. **Robot Base:** The robot base consists of a square with two wheels mounted on the front and two casters at the back. The uORCBoard and laptop are mounted directly on the robot base. A bridge containing the camera is at the front of the robot and the robotic arm is attached to the front of the robot base.
4. **Xtion Camera Feed:** An ASUS Xtion depth camera is mounted at the center of the bridge on top of the robot base and is used for block identification and collection as well as localization and wall following.
5. **Robotic Arm:** Attached to the front of the robot is a 2 degree-of-freedom robotic arm used for block collection. The arm is made of two servos attached to the shoulder and elbow. At the end of the arm is a gripper equipped with a servo to control grasping as well as a bump sensor for block detection, and rubber bands to hold blocks in place.
6. **Hopper:** The Hopper is the storage device for the blocks which organizes them into a 3 x 2 standing tower of blocks. The tower will be constructed by driving forward after six blocks have been collected and stored in the hopper.
7. **uORCBoard:** The uORCBoard is the microcontroller of robot. It is the communication unit between the software on the laptop and the robot's actuators and motors.
8. **Lenovo Thinkpad Laptop:** The laptop is used to perform all of the computation necessary for our implementation of the challenge and will be placed on the robot's base.

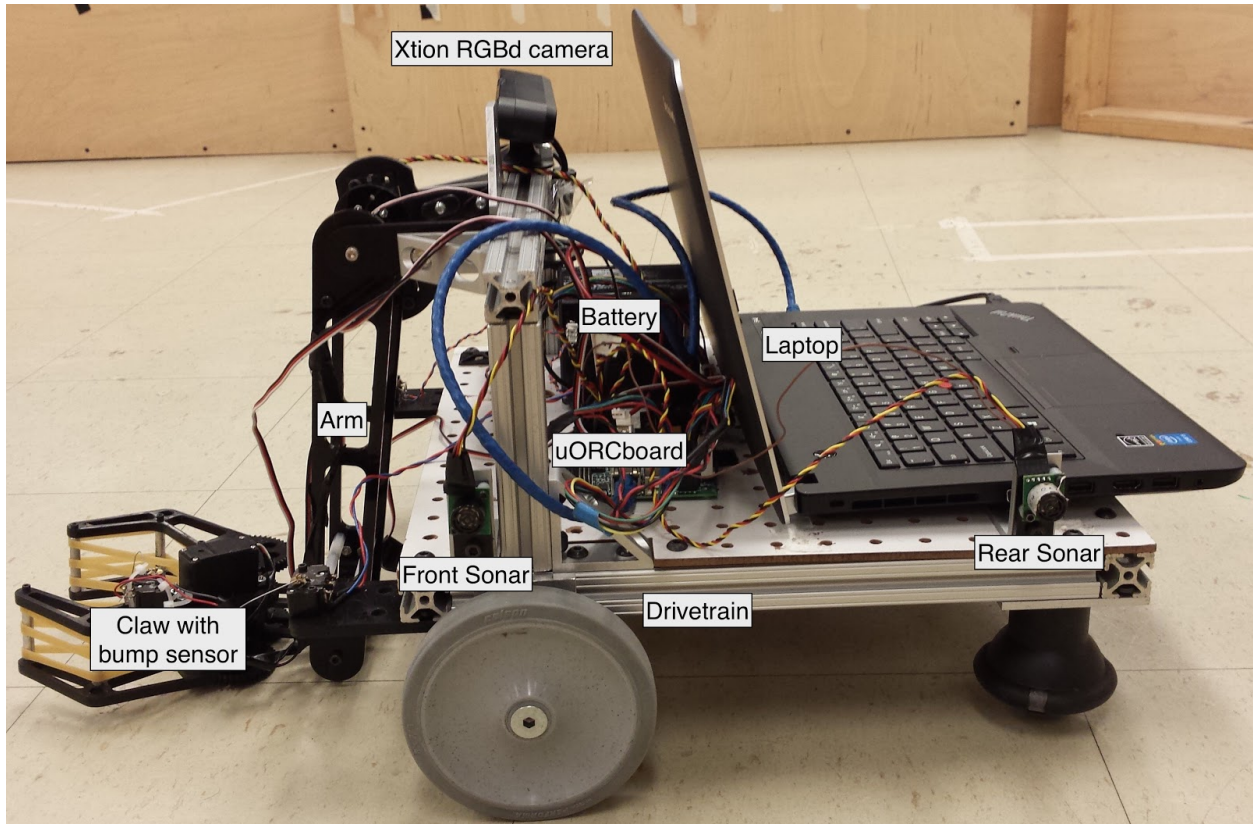


Figure 1. Labeled Diagram of Physical Robot without the Hopper

Overview: Software (Matthew and Daniel)

We describe two potential implementations of the software side of our approach to the challenge. We have labelled every subsection in this technical approach section with which implementation it belongs to. These implementations are:

1. **First Implementation:** An approach wherein the data from the robot odometry, Xtion depth and RGB images and sonars are combined to localize the robot's position within the given map. The robot uses RRT to plan its motions in the three-dimensional configuration space of the robot to explore the map and it collects blocks when they are first seen. Previously unidentified walls are incorporated into the robot's map using a basic mapping scheme.
2. **Second Implementation:** An approach relying almost exclusively on reflexive control. The data from the Xtion depth and RGB images and sonars are used to implement a reflexive wall-following method. The robot will collect blocks when they are first seen. If there is time to implement localization, we will use the odometry and localization to inform the wall-following method and make it not purely reflexive.

The first implementation is the primary implementation that we aim to produce. However, we recognize that it is an ambitious task to complete and have the first implementation working robustly. The second implementation is our contingency plan if we do not have the first implementation working in time for the challenge. The second implementation is designed to use a lot of the same code as the first implementation, with the major difference being the use of a simpler wall-following scheme rather than basic mapping and localization. In this section, we outline the technical details of both implementation plans and in the next section we outline the planning details of both implementations and what milestones we need to have achieved each week in order to remain with the first implementation or decide to change to the second implementation.

The software driving our robot will consist of several ROS nodes each running in a separate thread and each implementing a principal module of the code we will design for the challenge. The different ROS nodes we will implement are briefly described below.

1. **Xtion Camera Node:** This ROS node publishes the RGB image data and depth data read by the ASUS Xtion depth camera.
2. **Sensor Nodes:** These ROS nodes publish sensor data that will be read by the finite state machine and motion planning/drive nodes to determine our robot's motion. These sensors are each given individual ROS nodes and include the encoders, each of the bump sensors and each of the sonar sensors.
3. **Actuator Nodes:** These ROS nodes subscribe to messages from the hopper node and motion planning and drive nodes and signal actuators through the uORCboard. The actuators are given individual ROS nodes and include each of the wheel motors, each of the servos on the arm and each of the servos on the hopper.
4. **Odometry Node:** This ROS node subscribes to messages from the encoders and uses the data to estimate the coordinates (x, y, theta) of the robot on the map given an initial starting position of (0, 0, 0). It publishes these coordinates.
5. **Finite State Machine Node:** This ROS node subscribes to the sensor nodes and stores the current state of the robot along with the state transition logic that comprise the FSM used by the robot. The node publishes the current FSM state.
6. **Drive Node:** This ROS node subscribes to the current FSM state, the sensor nodes, the vision and image processing node, the odometry node and to the motion planning nodes. It publishes actuator values based on its state, the sensor readings and desired path determined at the motion planning node.
7. **Motion Planning Node:** This ROS node subscribes to the current FSM state, the sensor nodes and odometry node. It computes a desired path within the global map based on the odometry coordinates. It then publishes the path which is used by the drive node.
8. **Localization and Mapping Node:** This node subscribes to the Xtion camera node, the odometry node and the sensor nodes. The node localizes using this data within the prior map. If an obstacle clearly not matching any obstacle in the map is detected, then

the obstacle will be incorporated into the map. The node publishes the new obstacle data and the (x, y, theta) coordinates resulting from localization.

9. **Vision and Image Processing Node:** This ROS node subscribes to messages published by the Xtion camera node. It processes the camera image and publishes the key features of the processed image.
10. **Arm Node:** The ROS node subscribes to the current FSM state and actuates the servos controlling the arm when the robot is retrieving a block.
11. **Hopper Node:** This ROS node subscribes to the current FSM state and actuates the servos controlling the hopper mechanism accordingly.

The ROS message passing relationships between these key nodes are shown in the block diagram below.

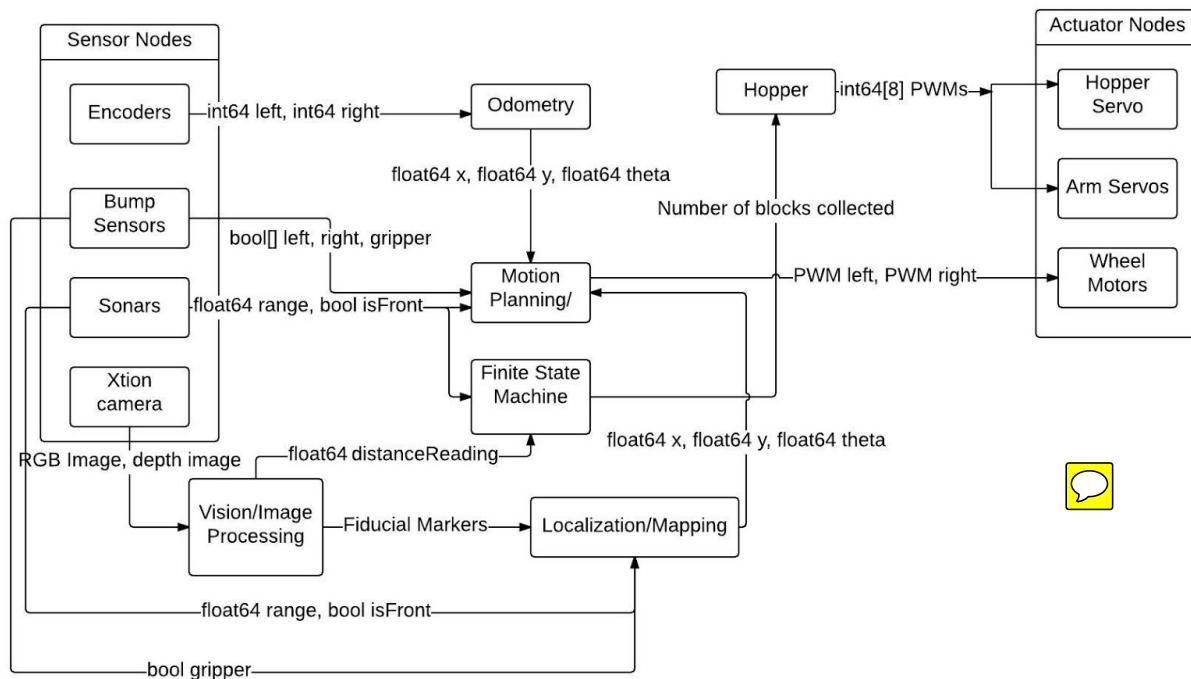


Figure 2. Message Passing Relationships Between ROS Nodes

The key modules among these nodes are the finite state machine, Xtion camera, vision and image processing, localization and basic mapping, motion planning, arm and hopper. The design decisions made in constructing these modules are described in detail in the sections below. Where applicable, we highlight the differences between these modules under the first and second implementations outlined above.

Finite State Machine (Matthew and Daniel)

In this section, we outline the finite state machines that we intend to use in both the primary and secondary implementations. The first implementation involves a smaller number of states

since the core of the implementation resides in the algorithms used for localization, motion planning and path following. Each of these tasks is implemented through a single algorithms that runs in a loop in a single state of the FSM. The second implementation has a more complicated FSM since the wall following algorithm is implemented mostly through the FSM, rather than in a single state.

FIRST IMPLEMENTATION: FINITE STATE MACHINE

Our robot will cycle between completing three major tasks: exploration, block collection and shelter construction. Throughout this entire process, the robot will be continuously localizing and implementing a basic mapping procedure. Below is an overview of the different sub-states used in each of these tasks as well as a diagram depicting the state transitions comprising the FSM of our robot.

1. **Exploration:** The robot uses RRT to explore the map. At any given point, if a block is detected in the camera image, the robot begins block collection.
 - a. **Compute Path:** A goal point which is sufficiently far from the set of previous goal points is identified in the configuration space. A path is computed using RRT to this goal point in the configuration space.
 - b. **Follow Path:** The robot follows the path using a PID controller to set its position (x, y, theta) to the next position along the path. If the distance between the robot and its next waypoint exceeds a certain threshold, the robot is deemed to have deviated from its path and moves to the state Strayed from Path. If the robot collides with an obstacle while trying to follow the path, the robot enters its collision state.
 - c. **Strayed from Path:** The robot recomputes a path from its current position to its goal using RRT. The robot moves to the Follow Path state with this new path.
 - d. **Collision State:** If the robot collides unexpectedly with an obstacle or if the fiducial markers identified by the Xtion camera and sonars do not match any of the particles in the particle filter, the robot enters a collision state. The robot backs up and turns to a random orientation. The particle filter used for localization is reset to have a uniform prior over particles selected from around the map.
2. **Block Collection:** The robot turns to face the block, drives towards it and picks it up using its arm. Once the block has been picked up, the robot restarts block collection if another block is in its field of view, continues exploration if there is no other block in its field of view and begins shelter construction if it has collected six blocks. One important note is that we do not intend to use the information in the prior map indicating the positions of blocks. Instead we plan to pick them up as they fall within the field of view of the Xtion camera.
 - a. **Align with Block:** Using the coordinates in the camera image of the center of the block, as determined by the robot's vision node, the robot uses a PID controller to center its field of view on the block while moving slowly towards it.



- While the tasks outlined above are being carried out, the robot is continually localizing using a particle filter within the (x, y, theta) three-dimensional configuration space of the robot. The details of this localization procedure are outlined in the section devoted to the corresponding module below. The state transitions for this FSM are outlined in the figure below.

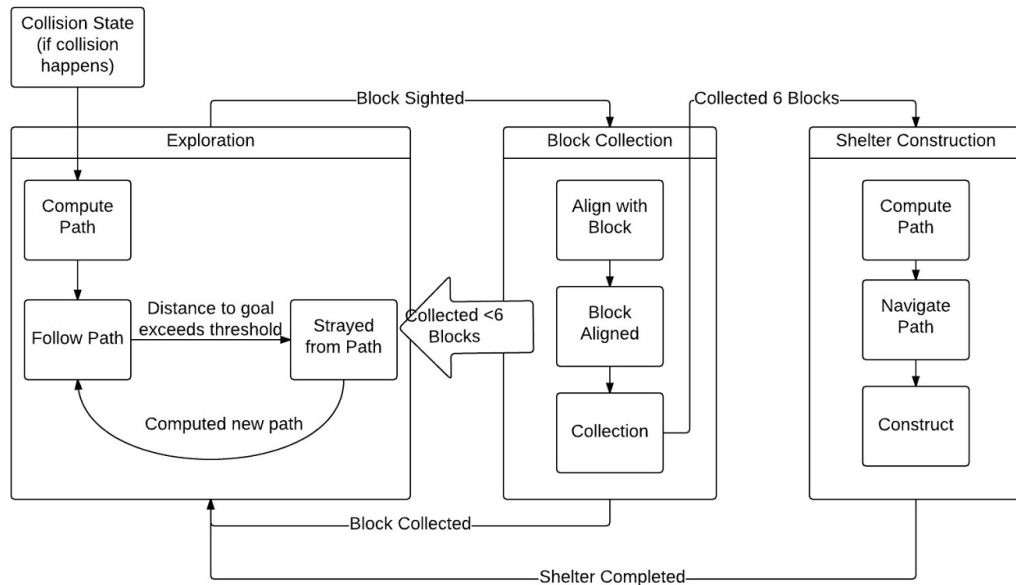


Figure 3. First Implementation Finite State Machine

SECOND IMPLEMENTATION: FINITE STATE MACHINE

In our second proposed implementation, our robot will cycle between completing three major tasks: wall following, block collection and shelter construction. Each one of these tasks is implemented using several sub-states in our FSM. Below is an overview of the different sub-states used in each of these tasks as well as a diagram depicting the state transitions comprising the FSM of our robot. The block collection and shelter construction tasks are similar to those in the first implementation with some minor changes.

1. **Wall Following:** The robot travels at a fixed distance on its left from the wall while aligned with the wall and avoiding obstacles. At any given point, if a block is detected in the camera image, the robot begins block collection.
 - a. **Find Wall:** The robot moves forward at a specified speed until a wall is either detected ahead or to the left of the robot.
 - b. **Align with Wall:** The robot moves forward while using a PID to align with the wall at a specified distance of approximately 0.2 meters from the wall to the left of the robot.
 - c. **Wall Ahead:** If the Xtion detects a wall is ahead, the robot rotates clockwise and in-place until a wall is no longer detected ahead and instead only to the left of the robot, in which case the robot will proceed to align with the wall.
 - d. **Turn at Wall End:** If the robot does not detect a wall ahead and ceases to detect a wall on its left, it rotates counterclockwise while moving slightly forward until it finds a wall.

- e. **Collision State:** If at any point during wall following, the bump sensors on the robot are triggered signalling a collision or if driving the wheel motors in opposite directions does not cause a change in the angular position of the robot according to the encoders, the robot enters the collision state. If the robot remains in this state for longer than a timeout, the robot will back up, turn in-place to face a random direction and then begin to find a wall again.
2. **Block Collection:** The robot turns to face the block, drives towards it and picks it up using its arm. Once the block has been picked up, the robot restarts block collection if another block is in its field of view, wall-follows if there is no other block in its field of view and begins shelter construction if it has collected six blocks.
- a. **Align with Block:** Using the coordinates in the camera image of the center of the block, as determined by the robot's vision node, the robot uses a PID controller to center its field of view on the block while moving slowly towards it.
 - b. **Block Aligned:** Once the block is sufficiently close to the robot and located in the center of the robot's field of view, the robot drives forward at a slow speed until the bump sensor in the gripper of the robotic arm is triggered.
 - c. **Collection:** The robot closes the gripper around the block, picks up the block and deposits it in the funnel leading to the storage unit at the back of the robot.
3. **Shelter Construction:** The robot computes a desired path to the shelter construction site, moves along this path while avoiding obstacles and then releases the 3 x 2 configuration of blocks stored in the hopper at the construction site.
- a. **Compute Path:** The robot does not move while it computes a C-Space path from its current position to the construction site using RRT.
 - b. **Navigate Path:** The robot moves from waypoint to waypoint along this path while avoiding collisions with obstacles detected with the front and side sonars.
 - c. **Construct:** The robot releases the 3 x 2 tower of blocks stored in its hopper at the construction site by opening the hopper, which will leave the rear of the tower unobstructed, and then driving forward and closing the hopper again.



The state transitions in the FSM described above are shown in the diagram below.

FIRST IMPLEMENTATION: FIDUCIAL MARKER EXTRACTION

For each depth image read by the Xtion, we will examine **N** lines of pixels above the horizon line of the image where **N** is between 3 and 5. From these lines, we will average the distance to pixels in several intervals and obtain approximately 3 or 4 averaged distance values along each of the **N** lines of pixels. Since these lines are above the horizon line, they correspond to averaged distances to walls. These averaged distances will be compared to the expected distances at each particle within the particle filter along with the expected sonar readings at each particle. These fiducials will be essential to the sensor update step of the particle filter.

SECOND IMPLEMENTATION: WALL DISTANCE EXTRACTION

For each depth image read by the Xtion, we will examine a single line of pixels above the horizon line. Along this line, we will average the distance to pixels in approximately 3 to 4 evenly spaced intervals. These averaged distance values correspond to the approximate distances to walls from 3 to 4 different angles from the focus of the camera. These will be used to detect whether there is a wall ahead, a wall on the left or a wall on the right. These distance values will be used directly in wall following in the second implementation.

FIRST AND SECOND IMPLEMENTATIONS: BLOCK RETRIEVAL

The vision and image processing node is used to compute the coordinates of the center of the closest block within the robot's field of view and approximate the distance between the robot and the block. This will be achieved through the algorithm outlined below.

1. Obtain a new image **IM** from the camera on the robot and apply a Gaussian filter to produce a smoothed version of the image **IMS**.
2. Using pre-tuned HSB-space cutoffs designed to recognize the potential block pixels based on their HSB representation, produce several binary images **B1**, **B2**, ..., **Bk** from **IMS**. The image **Bi** is nonzero at the set of pixels identified to be part of blocks of color **i** and is zero elsewhere. Note that here **k** denotes the number of block colors.
3. For each image **Bi**, apply the two-pass connected components algorithm to produce a discrete-valued image **Ci** and then determine the centroid of the connected component of largest area.
4. Report the closest block as the largest connected component with area exceeding a minimum threshold across all of the images **B1**, **B2**, ..., **Bk**. Report its area and the coordinates of its centroid in the field of view of the robot.
5. The distance **D** between the camera and the block can be approximated as

$$D = \frac{R}{\sin(A)} = \frac{R}{\sin(c \cdot P)}$$

where **R** is the approximate radius of the block, **A** is the angle formed by the block along the lens, **P** is the radius in pixels of the block and **c** is an empirically determined

constant such that $\mathbf{A} = \mathbf{cP}$. Note that such a constant has to exist since \mathbf{P} and \mathbf{A} are proportional.

The quantities discussed in Step 5 are depicted in the diagram shown below, where the block here is approximated as a sphere for calculation purposes.

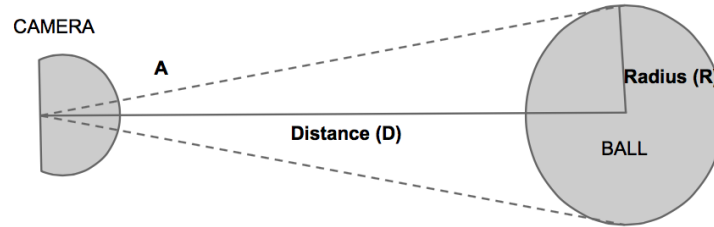


Figure 5. Camera-Object Relationships

We also will consider an alternative method of retrieving the distance to the closest block using the depth data recorded by the ASUS Xtion depth camera. This will be accomplished by using the distance recorded by the Xtion at the pixel corresponding to the centroid of the block. We plan to compare this method against the analytic solution described above for accuracy. In particular, we intend to use the Xtion solution at short distances and potentially the analytic solution at large distances to the closest block.

Particle Filter Monte Carlo Localization and Mapping (Matthew)

In this section, we outline the particle filter Monte Carlo localization method we intend to implement as well as the basic mapping method we intend to implement to account for walls that are present physically in the challenge region but not in the original map. These methods are outlined in the two sections below.

FIRST IMPLEMENTATION: MONTE CARLO LOCALIZATION WITH A PARTICLE FILTER

We implement a Monte Carlo localization method using a particle filter as described in *Probabilistic Robotics* by Thrun, Burgard and Fox. In particular, the localization algorithm involves the following steps used to compute a set of M particles $\mathbf{X}(t) = \{\mathbf{x}_1(t), \mathbf{x}_2(t), \dots, \mathbf{x}_M(t)\}$ at each time t where each particle $\mathbf{x}_i(t) = (x, y, \theta)$ is a possible position of the robot within the robot's configuration space. For each time t :

1. Iterate through the list of previous particles $\mathbf{X}(t - 1) = \{\mathbf{x}_1(t - 1), \mathbf{x}_2(t - 1), \dots, \mathbf{x}_M(t - 1)\}$ and for each particle, compute the expected location $\mathbf{x}_i(t)$ of the robot given that the robot resided at particle $\mathbf{x}_i(t - 1)$ at time $t - 1$ and given the encoder displacement $\mathbf{e}(t) = (\mathbf{eL}(t) - \mathbf{eL}(t - 1), \mathbf{eR}(t) - \mathbf{eR}(t - 1))$ of each encoder since time $t - 1$.
2. Iterate through the list of new expected locations of the particles $\mathbf{x}_i(t)$ and compute the probability $\mathbf{p}_i(t)$ that the robot resides at the position $\mathbf{x}_i(t)$ given the following fiducial markers:

- a. **Sonar readings at time t:** The most recent sonar readings at time t are compared to the expected sonar readings given that the robot is at position $\mathbf{x}_i(t)$ and this is used with the fiducial markers in part B to generate $\mathbf{p}_i(t)$.
 - b. **Xtion wall distances at time t:** The most recent Xtion wall distance readings at time t are compared to the expected sonar readings given that the robot is at position $\mathbf{x}_i(t)$ and this is used with the fiducial markers in part A to generate the probability $\mathbf{p}_i(t)$.
3. If the maximum value $\mathbf{p}_i(t)$ across all $i = 1, 2, \dots, M$ does not exceed a minimum threshold, then we assume that this indicates that there is an obstacle ahead not accounted for in the map. In this case, we set $\mathbf{X}(t) = \{\mathbf{x}_1(t), \mathbf{x}_2(t), \dots, \mathbf{x}_M(t)\}$ and apply the algorithm described in basic mapping to map the obstacle. In other words, we deterministically update the previous particles from time $t - 1$ using the encoder displacement values and not taking into account fiducials.
 4. If not Step 3, then we randomly sample the set $\mathbf{S} = \{\mathbf{x}_1(t), \mathbf{x}_2(t), \dots, \mathbf{x}_M(t)\}$ a total of M times such that the probability of selecting $\mathbf{x}_i(t)$ is proportional to $\mathbf{p}_i(t)$. The resulting set of particles forms the particles at time $\mathbf{X}(t)$.
 5. If Step 3 was carried out, the current localization $\mathbf{l}(t)$ is defined to be $\mathbf{x}_i(t)$ where the previous localization was $\mathbf{l}(t - 1) = \mathbf{x}_i(t - 1)$. In other words, in the case of step 3, we deterministically update the previous localization using the encoder displacement values.
 6. If Step 4 was carried out, the current localization is the particle $\mathbf{x}_i(t)$ such that its corresponding probability $\mathbf{p}_i(t)$ was maximal.

We plan to model the probability $\mathbf{p}_i(t)$ as follows:

$$p_i(t) = \left(\sum_{\text{Fiducials } f} (f_{\text{sensor}} - f_{\text{particle } x_i(t)})^2 \right)^{-\alpha}$$

In other words, we model $\mathbf{p}_i(t)$ as inversely proportional to the sum of the squared deviations between the fiducials measured by the sensors and the fiducial expected from the particle $\mathbf{x}_i(t)$. The value of alpha will be a positive real number which is experimentally determined to yield good localization results.

The fiducials expected from the particle $\mathbf{x}_i(t)$ will be predicted from the map and position $\mathbf{x}_i(t)$. For example, the expected sonar readings can be determined by the length of the segment in the map adjoining $\mathbf{x}_i(t)$ and the intersection points of lines perpendicular to its orientation with obstacles on its left. Similarly, the expected values of Xtion distances to walls can be computed as the lengths of the segments intersecting the walls ahead of the robot.

FIRST IMPLEMENTATION: BASIC MAPPING

As indicated in the localization section above, when the particles from the particle filter do not have a high enough maximum probability under the model described above, we will deem

there an obstacle ahead that has not yet been mapped. In order to map this obstacle, we fit a line segment to the obstacle.

To do this, we use the fiducial markers extracted from the Xtion depth image. We fit a linear function to the extracted fiducials using the following algorithm:

1. Let **D1, D2, ..., Dk** be the extracted distances to the walls ahead along some line of pixels above the horizon line in the Xtion depth image.
2. Begin fitting (with minimum mean squared distance) a linear function of the form $f(i) = ai + b$ to the distance **Di**. Stop when **Di+1** is sufficiently far from $f(i+1)$ where f is the linear function fit to **D1, D2, ..., Di**.
3. Begin fitting another linear function starting at **Di+1** and continue this process until all of the points **Di** are fit to exactly one linear function.
4. Suppose that the linear functions produced are **f1, f2, ..., fm**. Map the segments corresponding to each of these functions evaluated over the ranges of i for which they were fit onto the map. Map these functions only if they are sufficiently far from obstacles already on the map.

This enables multiple obstacles within the field of view to be mapped at once. This is useful for cases in which a missing obstacle has multiple sides which are within the field of view. Combining this with the localization algorithm described above yields a method for particle filter localization and basic mapping that works effectively given the assumption that the prior map is correct other than with respect to omitted obstacles.




Motion Planning and Exploration (Matthew)

Motion planning will be carried out for two purposes: (1) exploration and (2) navigation to the construction site to release the shelter. The algorithms that will be used for each of these two purposes are outlined below. We also outline an extension of the second implementation that includes a map-informed wall following method. We would only implement this extension if we: (1) are not able to implement the first implementation; and (2) have additional time after completing the second implementation and successfully implement localization. In describing this section, we assume that we have successfully localized using a particle filter as outlined in the section above and have obtained a reasonable set of coordinates (x, y, θ) for the robot.

FIRST IMPLEMENTATION: MOTION PLANNING FOR EXPLORATION

Motion planning for exploration is carried out in the first implementation as an alternative to wall following. It consists of the following steps, which are called each time the robot finishes collecting a block or constructs a shelter:

1. A point $\mathbf{G} = (G_x, G_y, G_{\theta})$ in the configuration space approximately minimizing the sum $\|\mathbf{G} - \mathbf{P}\|$ over all points \mathbf{P} that have previously acted as goal points is first identified. We plan to find \mathbf{G} by randomly sampling the configuration space several times and choosing the point with the lowest distance sum as described above.
2. If the current output of the particle filter localization algorithm is $\mathbf{Q} = (x, y, \theta)$, RRT is applied to identify a configuration-space path from \mathbf{Q} to \mathbf{G} . 
3. The robot will navigate along the path using a PID controller on the robot's rotational and translational velocity to converge towards the desired waypoints $\mathbf{W}_i = (x_i, y_i, \theta_i)$ along the configuration-space path. If the robot strays too much from this path, by a distance of at least 0.2 meters from the desired waypoint, then RRT will be applied again to identify a new path to the goal point \mathbf{G} from the current position.

FIRST AND SECOND IMPLEMENTATIONS: MOTION PLANNING FOR SHELTER CONSTRUCTION

Motion planning will be carried out after six blocks have been collected and assembled in the hopper into a 3 x 2 tower. The steps in this procedure are very similar to those described above with several changes. These are outlined in the steps shown below.

1. A point $\mathbf{G} = (G_x, G_y, G_{\theta})$ in the configuration space representing the construction site is identified either as a given point or as a point maximizing its amount of surrounding obstacle-free space on the map.
2. If the current output of the particle filter localization algorithm is $\mathbf{Q} = (x, y, \theta)$, RRT is applied to identify a configuration-space path from \mathbf{Q} to \mathbf{G} .
3. The robot will navigate along the path using a PID controller as described above along the configuration-space path. If the robot strays too much from this path, by a distance of at least 0.2 meters from the desired waypoint, then RRT will be applied again to identify a new path to the goal point \mathbf{G} from the current position.
4. On arriving at the construction site, the robot will release the hopper and drive forward for a specified amount of time until the 3 x 2 tower is standing alone. The robot will then return to the exploration phase of motion planning.

SECOND IMPLEMENTATION: MAP-INFORMED WALL FOLLOWING

If we change to the second implementation and time permits, we also plan to implement a more robust wall-following algorithm using motion planning and localization. For wall-following, we plan to use a PID controller based on a potential function that combines map information with sonar readings when wall-following. This potential function would penalize close distances to walls detected by the sonars and X_{tion} as well as distances from the localized coordinates (x, y, θ) to walls on the map. For example, this potential function could be the sum of the signed differences between the actual and desired distances to the walls as determined by the localized coordinates (x, y, θ) on the map, the sonar readings and the X_{tion} . We would then use a PID controller on the motor velocities based on the error

computed using the potential function. This wall following approach could yield a smoother wall-following mechanism and provide robustness against sensor reading noise or sensor failure.

Arm (Ryan and Daniel)

Our robot will use the arm from Lab 7 to grasp objects and place them into the hopper. It will visual-servo to the block, move the arm with the gripper open to a grasping position, and move forward until the block enters the grasping mechanism. Once the robot has the block, the gripper will close, the arm will lift the block and move it to a specific position above the hopper, at which point the gripper will release the block.

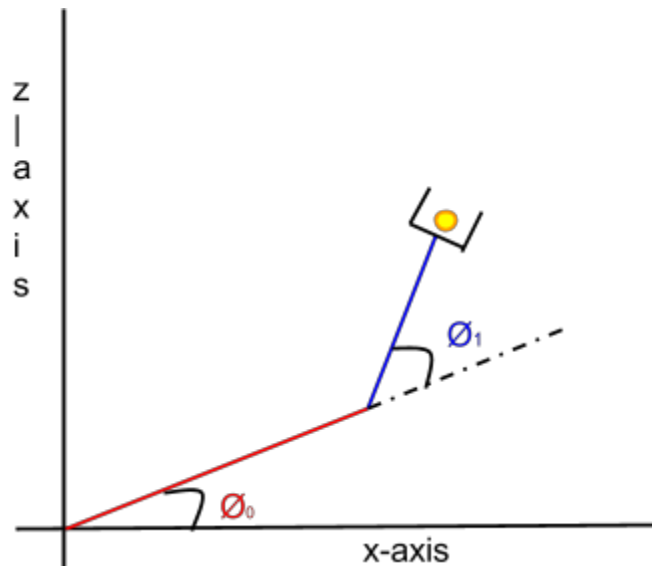


Figure 6. Two DOF Robotic arm

In the figure above, the red link is manipulated by the shoulder servo, the blue link is manipulated by the elbow servo, and the gripper is attached at the end. The robot has two predefined angles θ_0 and θ_1 . θ_0 is defined from the ground to the red link and θ_1 is defined from the red link to the blue link. The yellow dot in the figure is the end-effector, the location we want to place the blocks for grasping.

Using inverse kinematics, the robot takes the approximated (x,z) coordinates of the blocks from visual-servoing and converts them into appropriate angles (θ_0, θ_1) for the servos. However, the inverse kinematics of this system will produce two solutions for most of the (x,z) inputs, so the extra constraint of keeping θ_1 has been implemented in order to allow for one solution configuration. The constraint and inverse kinematics are below:

$$\theta_0 = \tan^{-1}(z/x) - \cos^{-1}((x^2 + z^2 + L_1^2 - L_2^2)/(2*L_1*\sqrt{x^2 + z^2}))$$

$$\theta_1 = \pi - \cos^{-1}((L_1^2 + L_2^2 - x^2 - z^2)/(2*L_1*L_2))$$

Figure 7. Inverse Kinematics

$$\theta_1 > 0$$

Figure 8. Constraints

Hopper Design (Erin and Ryan)

The robot will utilize a passive storage unit to organize the blocks into the correct configuration as they arrive, as pictured in the diagram below.

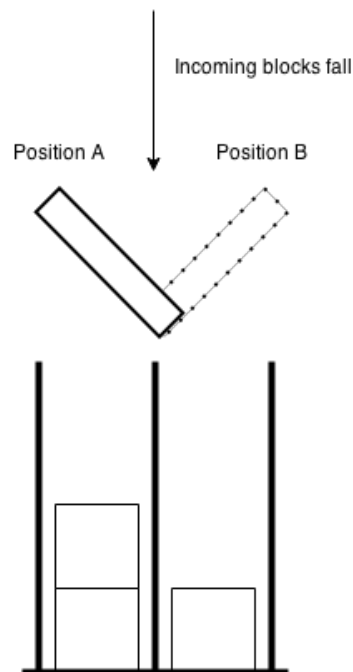


Figure 9. Hopper Design

When each block is released by the gripper, it will fall down a central chute. A simple rotating flipper will be positioned at the bottom of the chute. A servo will drive the flipper to one of two possible positions, as shown in the diagram. When the flipper is in position A, the block will fall to the right side of the hopper. When the flipper is in position B, the block will fall to the left side of the hopper. A stack of blocks will accumulate in each side of the hopper. A central divider inside the hopper will prevent blocks from toppling from one side to the other as the robot moves.

RELEASING BLOCKS

At the end of the challenge, the robot will release the blocks in a specified location and drive away. The hopper will have an open bottom, so each stack of three blocks will rest on the floor and be dragged along as the robot drives around. This allows blocks to be in an easy position for release at the end without added complexity of dropping blocks from atop a mechanism that holds them up. To manage the release of the tower, the back wall of the hopper will be a hinged door, which will be controlled by another servo similar to the switch at the top of the hopper. During the block collection phase, this door will be closed. When the FSM determines that a tower is ready to be deployed, the robot will simply open the door and drive forward, leaving a 3 x 2 tower of blocks behind.

SERVO CONTROL

As described above, there will be two servo motors controlling the function of the hopper at the back of the robot - the switch determining which side of the tower blocks will be added to, and the latch mechanism which can open and close the back door.

Throughout the challenge, the switch servo will be triggered to rotate each time the arm collects a block and dispenses it in the hopper. The bump sensor inside the arm's grabber will indicate when a block is collected, and the servo code will instruct the motor to switch sides a few seconds after the bump sensor indicates that the block has been released.

The back door servo will only ever be triggered in two situations during the challenge - once a full tower of 6 blocks has been constructed in the hopper, or at the very end of the round, a few seconds before the time ends. The FSM will keep track of how many blocks have been picked up and deposited in the back by using the bump sensor inside the claw, and it will initiate a deploy sequence when that value reaches six blocks. At the end, regardless of if a new tower is complete at the end of the round, it will be deployed in whatever state it finishes in.

Alternate Mechanical Design (Erin)



If our hopper is not assembled and working reliably by the third milestone, then we will implement a much simpler block collection mechanism. Instead of using the arm to deposit blocks into the hopper, we will drive up to blocks and drive over them so that they move beneath the robot into a walled region. At the end of the challenge, we will lift the rear wall of this region and drive forward, leaving behind a 1-block high wall of blocks that have accumulated under the robot. This would yield a much simpler block collection procedure and a mechanically simpler shelter construction procedure.

Calibration and Testing Procedures (Daniel and Ryan)

The calibration and testing procedure of the robot are run independently in order to catch small errors in the individual modules before compiling the modules together for a grand run. This section focuses on testing procedures, but not the failing protocols. Those can be found in the risk mitigating section.

A key factor in the calibration procedures is the participation of a human operator. These procedures will prompt the user requesting that certain actions be performed, or that certain real-world physical measurements be entered. For programmatic constants that are not subject to change (such as servo positions), calibration programs need to be run just once, to populate the appropriate values in a file containing all the system constants. For constants that are subject to change (such as lighting constants), the calibration procedures will be run each time there is a relevant change to the robot or its environment.

SONAR CALIBRATION




Sonar Calibration will be done by taking different readings of the two side sonars at different distances from the wall. Specifically, the program will repeatedly prompt the user for the precise, measured distance to the wall, and compare that to sonar readings. From the readings, we can establish a conversion factor from meters to sensor value and an ideal range of what we should be reading. This will help us design a filter to exclude outlier readings that can cause brief extreme behavior for the robot.


VISION CALIBRATION

Similar to sonar calibration, vision calibration will accumulate distance readings in order to create a conversion factor and an outlier filter. The program will prompt the user to place blocks of various known colors in a certain position in the camera's field of view and use that to determine the proper calibration constants. Vision calibration also includes adjusting machine vision filters in order to identify blocks and how far away they are. Again, an iterative user feedback will be crucial in determining the proper parameters; in this case, the user will be able to tweak parameters on the fly, as the program runs, and observe which set of parameters works best. Once the filters are complete, a brief Visual Servo test will be run to ensure that the robot can identify a block, move to it, and pick it up.

SOFTWARE TESTING

The software testing is split into 3 different modules of the robot; the FSM, motion planning, and localization.

Finite State Machine  The FSM of the robot will be tested via software simulation of the challenge where each state will be tested on input, output, and transitions. In general, the test will assume a state and given certain inputs, check to make sure proper transitions are being made. There will be both “clean” input data sets, that comprise the values we expect in an ideal world. We will also perform simulations using empirically gathered data, including noise and all. Once the state machine passes the simulations, the three modules, wall following, block collection, and shelter construction, will be checked with test runs focusing on each independently. Block collection and shelter construction will be covered by the arm and hopper test procedure, while the wall following will have tests runs similar to lab5 in class. Each run will keep track of the inputs from sensors, outputs to the robot’s actuators and a current state in order to make sure actions taken by the robot are following the design.

Motion planning: One procedure will produce a set of waypoints based on the robot’s current location, the map, and a goal point. This list of waypoints will be displayed to a GUI in an intuitive way, just as in Lab 6. This will allow us to debug any issues with our waypoint selection algorithm. 

A second procedure will test the robot’s ability to follow perform waypoint navigation. At each time step, the program will record the distance between the robot’s current location and the path it is supposed to be following. By measuring the accuracy of waypoint navigation, we can pinpoint the source of issues. For example, if the robot’s real-world behavior is errant, we can check if the waypoints seem correct. If they do, we can assess if the robot’s failure to follow that correct path is due to poor waypoint navigation. If the waypoint navigation procedure indicates that the robot is on track, then perhaps our problem is poor localization. Note that this waypoint navigation accuracy test will be designed such that it can run with assistance of different localization schemes. For example, it can rely on only odometry and initial position for localization, as in Lab 6. It can just as easily be reconfigured to use our particle-filter localization instead.

Localization: There will be a low level and a high level of testing for our particle-filter localization. At a low level, unit tests will demonstrate the correctness of the probability function $p_i(t)$ that takes in a set of fiducials and outputs the probability that the robot is in a certain location. At this stage, we will play around with different locations (gathering fiducials based on real, not simulated, data input), and tinker with the value of alpha until we’re happy with the performance. That is, we’d like to see the program output a high probability at one location (the truly correct one), and low probabilities at the rest. This metric is subjective, and as such our calibration procedure is subjective.

At a higher level, we’ll write a procedure that allows for manual control of the robot via keyboard input. This procedure will also output the localization algorithm’s resulting position at each time step in a clear way. By manually driving the robot around, and observing the efficacy of the localization algorithm as we go, we can diagnose problem cases, and develop

strategies to mitigate those problems. Experimenting with different values of alpha may help improve our results.

ARM AND HOPPER TESTING

The testing procedure of the hopper is split into two parts; hardware construction to test the relative effectiveness of the robust design and software configurations to test the servo logic and deployment.

The hardware tests focus on the effectiveness of the design. The hopper passes if blocks can successfully fit in the storage unit, the latch door doesn't open due to frictional forces, and the switch provides enough influence in order to sort effectively. These tests are being performed on materials and theoretical models before the assembly of the hopper in order to save materials.

The software test focuses on calibrating the switch servo delay, the arm drop point, and the latch release mechanism. The test will be done by calibrating each servo's PWM with angles and then test the switch motion. Once calibrated, the arm will drop blocks at multiple points on the funnel to find a suitable drop range while the switch delay is calibrated for the drop zone. Deployment tests will run with different loads in the hopper unit, such as having, no blocks, only 1 tower of blocks, uneven towers, and the ideal 3 by 2 case.

MILESTONES AND IMPLEMENTATION PLAN

Overview: Implementation Plan

Our plan is to try to get done convex wall-following and our mechanical design by week 1, concave wall-following and machining by week 2, the final robot built and wired and block collection by week 3, shelter construction and the final challenge code by week 4, and everything debugged and potentially the more advanced wall-following algorithm and localization by week 5. The precise deliverables we have set for each week are outlined in the milestones section below.

Risk Mitigation

We have devised several contingency plans if the more ambitious parts of our design do not end up working. We have multiple fallbacks and failsafes planned in case of a failure to get something working reliably on either the software or hardware side of our design.

If we encounter issues with our funnel and hopper design, we will ditch them along with the arm in favor of the under-robot single-block-high wall collection scheme described in the Alternate Mechanical Design section above.

As far as software, we are planning to implement Xtion-based localization and RRT motion planning for navigation around the map during exploration, as outlined above as “First Implementation.” If the rest of the First Implementation is sound, but our RRT motion planning algorithm is not working by the fourth milestone, we will instead use the discretized BFS search algorithm that we implemented and got working reliably in Lab 6. If we run out of time or otherwise fail to implement the rest of the First Implementation, we will forgo the SLAM functionality in favor of a smart, advanced wall-following algorithm using a potential function outlined above as “Second Implementation.” If we do not have either method working by the second last milestone, we will fall back on a purely reflexive wall-following scheme.

Individual Responsibilities

The major parts of the implementation process that we are each responsible for are outlined in the table below. These were allocated to ensure that we can parallelize the implementation process as much as possible. We plan to have two people devoted to any major task at each time and for two tasks to be worked on at each time.

Team Member	Implementation Responsibilities
Daniel	Implementing the FSM, design of controls in wall-following, wiring the hopper and additional sensors, motion planning, localization, debugging the final code and tuning parameters
Erin	Building the hopper, wiring the hopper and additional sensors, design of controls in block collection, vision and image processing, debugging the final code and tuning parameters
Matthew	Implementing the FSM, vision and image processing, motion planning, localization, debugging the final code and tuning parameters
Ryan	Design of the hopper, building the hopper, design of controls in wall-following, design of controls in block collection, debugging the final code and tuning parameters

Milestones

Our implementation goals for each week are shown in the table below. We currently include our plan to implement the more advanced wall-following procedure outlined above and the localization method outlined above.

Deadline	Implementation Milestones
April 8 (Milestone complete)	<ol style="list-style-type: none"> 1. Set up Java source files for the FSM, motion planning and drive, vision and image processing and hopper nodes. 2. Decide on front sensor scheme (2 new sonars vs. Xtion). 3. Program and test the wall-following code in the FSM and have wall-following along convex obstacles working reliably. 4. Mechanically design the hopper and funnel.
April 15 (Milestone complete)	<ol style="list-style-type: none"> 1. Port over motion planning, vision and image processing code from Labs 3, 4 and 6 into the new source files. 2. Install Xtion PRO camera and integrate into code. 3. Get Lab 3/4 Visual Servoing working with new Xtion setup. 4. Complete CAD designs of funnel and hopper.
April 22	<ol style="list-style-type: none"> 1. Build the funnel and hopper and attach it to the back of the robot. 2. Complete assembly and wiring of full mechanical structure. 3. Add additional logic to wall following to implement the turn at wall end state. 4. Test wall following and have wall-following along concave and convex obstacles working reliably. Also develop transition logic in the wall-following code to ensure that transitions between the different states do not trigger accidentally. 5. Revise the implementation of vision and image processing from Labs 3 and 4 to implement the vision algorithm outlined above. 6. Complete basic functionality of wall-following, visual servoing, and arm functions in code. 7. Stretch Goal: Partial functionality of motion planning / localization
April 29	<ol style="list-style-type: none"> 1. The robot should be completely built, able to wall follow, able to collect blocks separately and the motion planning, localization and mapping code should compile. 2. Implement motion planning, localization and mapping as described in the technical approach section above. 3. Implement hopper code. 4. Have the state transitions and overall block collection procedure beginning by identifying a block in the field of view until depositing the block in the funnel working reliably. 5. Test and have the shelter construction states of the FSM working reliably.

	<ol style="list-style-type: none"> 6. Integrate the code for the three main subtasks (wall-following, block collection and shelter construction) to produce the final code for the challenge. 7. Tune parameters and debug remaining issues with the code. 8. Stretch goal: construct shelter in a specified location. Use the Lab 6 code to implement the motion planning system used in shelter construction.
May 6	<ol style="list-style-type: none"> 1. The robot should be able to successfully complete the challenge. 2. Have tested and calibrated the motion planning, localization and mapping algorithms from the previous milestone. 3. Tune parameters and test the overall challenge routine on a variety of wall configurations, maps and block arrangements. 4. Stretch goal: Implement the more advanced wall-following algorithm described above and test it (if changed to the second implementation). If it performs better than the wall-following currently being used, add it to the final challenge code.

SELF-ASSESSMENT

Team Decision-Making Processes

In making decisions in preparing the CDD, our team proposed ideas in a group meeting until we arrived at an approach everyone was content with. Because our approach to constructing the robot's software uses similar ideas to those that we implemented in the seven labs, we often found ourselves agreeing on the software side of the design more readily than the mechanical side. Many of the software decisions we came across we had already considered and proposed a solution to while implementing the seven labs. The most time consuming part of our team's decision-making process was agreeing on a design for the hopper and ball collection mechanism. This may be due to the fact that there was no lab where we had to consider using a hopper whereas most other components of the robot's design we had at some point considered before.

Self-Assessment: Matthew Brennan

The collaboration that produced this CDD and which went into the labs we finished before Spring break has caused me to change many of the plans I outlined in my CDE. The mechanical side of the class is the most challenging for me. I realized that the mechanical plans I outlined in my CDE were infeasible given the materials available in the class. I planned to use a conveyor belt to move around the blocks, falsely thinking that this might be an easier task than using the robotic arm. After having received feedback from my time and after the experience of the seven labs, I now realize that using the robotic arm to move blocks around

the robot is a much more feasible task. I also did not have any background in motion planning on entering this class and, in my CDE, did not plan on using the motion planning methods I learned in lecture and that we implemented in Lab 6. I now realize that these are much more robust than the methods I proposed to use in my CDE. In outlining the CDD, the importance of having teammates to run ideas by was strongly reinforced to me.



From writing part of the technical approach section of this CDD, I have gained a much more concrete understanding of the details of integrating the pieces of code we wrote for the seven labs to produce the final code for the challenge. I also think I have a much better appreciation for the potential problems we might run into after thinking about risk mitigation and having run into similar problems while implementing parts of the seven labs.

I think my greatest challenge going into this final project will be on the hardware and electrical side of building the robot, which are my weakest areas. I also think a large challenge for the group will be debugging integration issues, i.e. problems that arise when two modules are integrated together but not when they are used separately. I think this will be one of the greatest challenges going from the lab code to our final challenge code.

Self-Assessment: Erin King

I approached the Grand Challenge for RSS with the mindset I have gained from participating in my high school robotics team for four years, and then in MASLAB this January. Because MASLAB was also a four-member team tasked with building a robot in just a few weeks to retrieve and stack blocks, I felt like I had an advantage coming into the planning process. However, I kept my CDE proposal very broad, not specifying a particular mechanical design. The few things I did specify included standard visual servoing and high-level control topics, as well as arguments to assemble our final structure on-board the robot to deploy all at once and to prioritize getting a working robot to accomplish the challenge using simple wall-following before attempting to implement complex localization and mapping techniques. My teammates all agreed on these points, and they will remain a focus in our final approach to the challenge.

Although after my previous robotics experience I had a bias towards building a familiar conveyor-style collection mechanism, my opinion was swayed in Lab 7 when we operated the robot arm and found it a reasonable and robust method of picking up blocks. Although I thought it was more complicated and slow to use an articulating arm at first, I realized that it solves jamming problems that I've faced with conveyor mechanisms in the past. I believe that our planned design for this year's Grand Challenge will be successful.

In all of the RSS lab assignments so far, I have been exposed to techniques in the mechanical, electrical, and programming aspects of robotics. I am looking forward to using the Grand Challenge as an opportunity to apply all of these skills, and hope to be a versatile member of the team rather than just working on one assigned subsystem.



Self-Assessment: Ryan King-Shepard

I entered 6.141 with the knowledge of control systems, software design, and hardware kinematics. During the course, I have been exposed to a variety of new technical topics such as visual-servoing, vision and image processing, and mapping with C-space. The course has also taught different techniques to deal with group conflict, organization, and presentation. The most effective was the technique of handling another person's issue toward a goal, which would have served me well in the past and hopefully be effective in the future.

I believe I do well in generating ideas, but fail to communicate or act on them in a timely manner. I also overthink challenges, forcing myself to learn every detail of the project. I don't usually work well with black boxes, which made lab 2 particularly challenging since I couldn't grasp the basics of the already developed module. However, experience in the class has allowed me to specialize toward my duties but still grasp the general idea.



On a communications level, I have improved when it comes to communicating technical effectively and have developed a rough process for group presentations. Although I still face confidence issue from time to time.

Self-Assessment: Daniel Mendelsohn

I think my biggest advantage is my technical background. I have a diverse skillset that I've acquired through many lab classes at MIT, and through a variety of projects that I've done. In 6.115, for example, I learned basic principles of analog circuits, digital circuits, signals, kinematics, and low level processing. I look forward to bringing these skills to bear on the RSS Challenge.

With that said, my main issues center around focus and motivation. In order to continue meeting deadlines, I'll have to become better at breaking complex tasks down into more manageable ones. In that light, our rigorous planning process is certain a blessing and not a curse, as it will allow me to keep myself accountable on a day-to-day basis. My teammates and I promote a culture of general accountability, and mutual support. I'm hopeful that they will help me maintain my motivation when it becomes depleted, and I will do the same in turn.

More than any project that I've done before, the RSS Challenge will challenge me to trust and rely on my teammates, and to become worthy of their trust. My greatest achievements at MIT have been accomplished by myself (such as my 6.115 final project). I know my group consists of a set of incredibly talented individuals, and I will need to have faith in their decisions, both technical and managerial.



CONCLUSION

After having thought out the design of our final robot and decided on the implementation outlined in this CDD, we feel prepared to begin the RSS Grand Challenge. We believe that our proposed solution will be relatively robust, and that it will allow us to prioritize finishing the challenge. We also believe that the distribution of responsibilities will allow us to parallelize work on the robot and achieve a relatively good level of efficiency. We plan on allocating a large amount of time to this and also recognize that many aspects of implementing our approach will take longer than we predict they will.