CuBi: Room Decluttering Robot

MRSD Team D

The Team:

Laavanye Bahl

Paulo Camasmie

Jorge Anton Garcia

Changsheng Shen (Bobby)

Nithin Subbiah Meganathan

Mentor:

Dr. David Held

Customer:

Cyert Center for Early Education

Carnegie Mellon University
December 2019

Abstract

Despite robotics making advancements in various fields, with the exception of iRobot's Roomba, no other robot has survived the home setting. It is because the Roomba is the only commercial robot that addresses a utility need. Our goal is to create a utility robot that declutters the floor. Families with both parents working find it laborious to maintain a clean home with their kids cluttering the floor with toys. For our customers, at Cyert daycare, teachers find it hard to clean the clutter while taking care of the babies. To tackle this problem we are developing CuBi - an autonomous robot that seamlessly declutters the floor off toys.

At this phase of the project, CuBi can perceive toys on the floor based on size, move towards it, pick it up, and drop it in the start position. It can perform this operation continuously until all the objects within a certain area are cleared of toys.

Table of Contents

1. Project Description

There is a need for a task-specific domain robot, that will help people with daily chores, such as the "Roomba" [1], a robotic vacuum cleaner. However, to our knowledge, there is no consumer-level robot that will declutter the room, a task that is needed prior to a vacuum operation. A mobile base with a robotic arm on it is not a novel concept. There are examples from toys to research-grade robots as shown in Figure 1. However, by optimizing the design of a robot, its mechanisms, perception, and algorithms to a specific task, it should be possible to develop an efficient, affordable, and commercial version of it [2][3]:

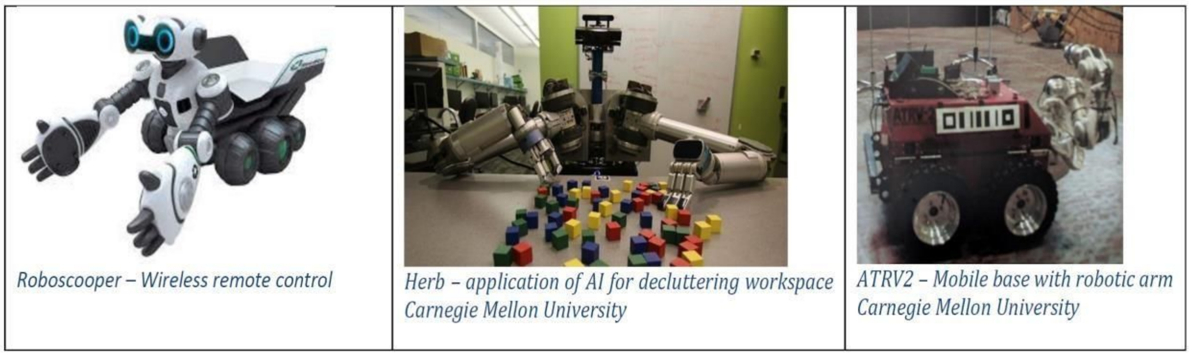

Roboscooper – Wireless remote control
Herb – application of AI for decluttering workspace
Carnegie Mellon University
ATRV2 – Mobile base with robotic arm
Carnegie Mellon University

Figure 1: Robot examples

The main goal of this project is to automate the task of picking up clutter to improve the daily lives of parents, pet owners, and daycare workers. By the end of the project, as shown in Figure 2, our robot should be able to encounter a room in an initial state, such as the one on the left, and work autonomously, avoiding people, pets and obstacles along the way, to achieve the state on the right, in an optimized matter, with all objects picked up from the floor and placed at the desired destination.

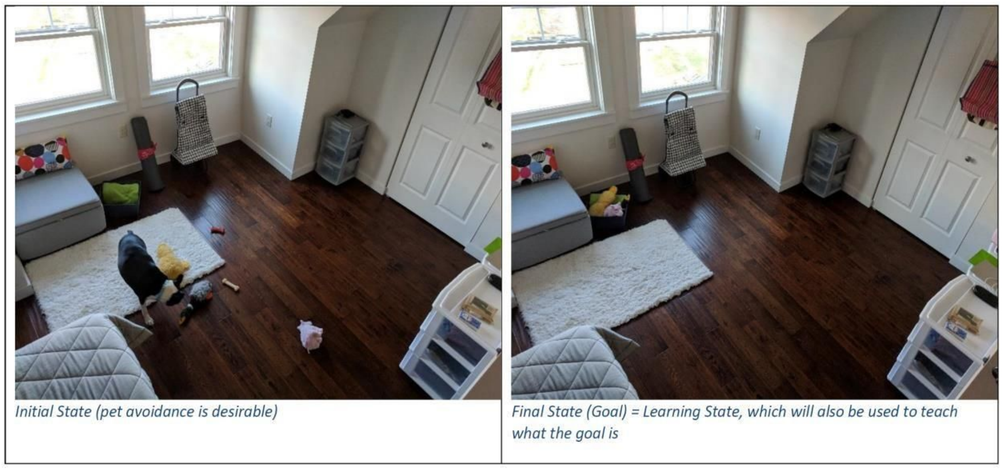

Initial State (pet avoidance is desirable)
Final State (Goal) = Learning State, which will also be used to teach what the goal is

Figure 2: Project goal

2. Use Case

Zachary is an early childhood educator in an infant-toddler classroom. Most of the children are under one year of age. Ellen, who is under his care, suddenly started crying after dropping a rattle from her hand. As Zachary comforted her, he looked around the room. Over the last hour, the room had become cluttered with a dozen toys. As the children explored the space, the educators were busy with changing diapers, giving bottles, and offering children snacks. Now that all of that is finished, it was time to get the room cleaned up and organized so the morning activities can begin.

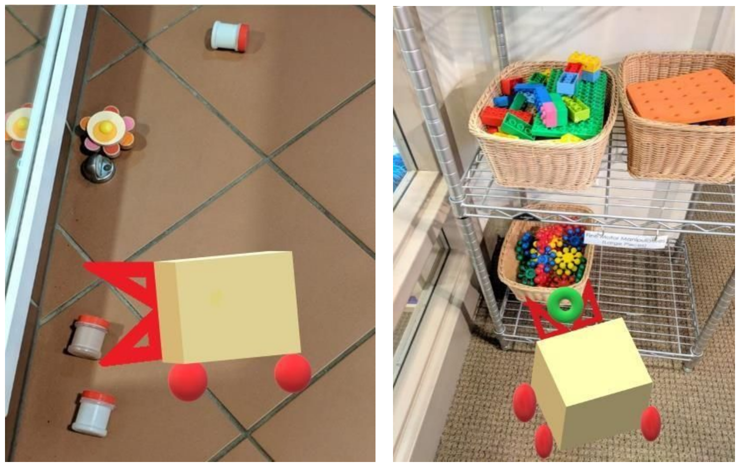


Figure 3: CuBi's desired operation

Once Ellen stopped crying and was playing peek-a-boo with an educator in the other playroom, Zachary turned on CuBi to perform its work while he gathered the materials needed for the light and shadow exploration. CuBi went around the room, when no babies were around, picking up tennis ball-sized toys from the floor and placed them in the bin at the corner of the room as shown in Figure 3. For 30 minutes, CuBi performed its work without crashing into anything and placed almost all the toys in the bin. Then CuBi went back to its dock to self-recharge for a few hours. Now that the floor was decluttered, Zachary was ready to set up for the light and shadow activity. He was happy to see the clean floor as shown in Figure 4 and quickly set up the next experience.

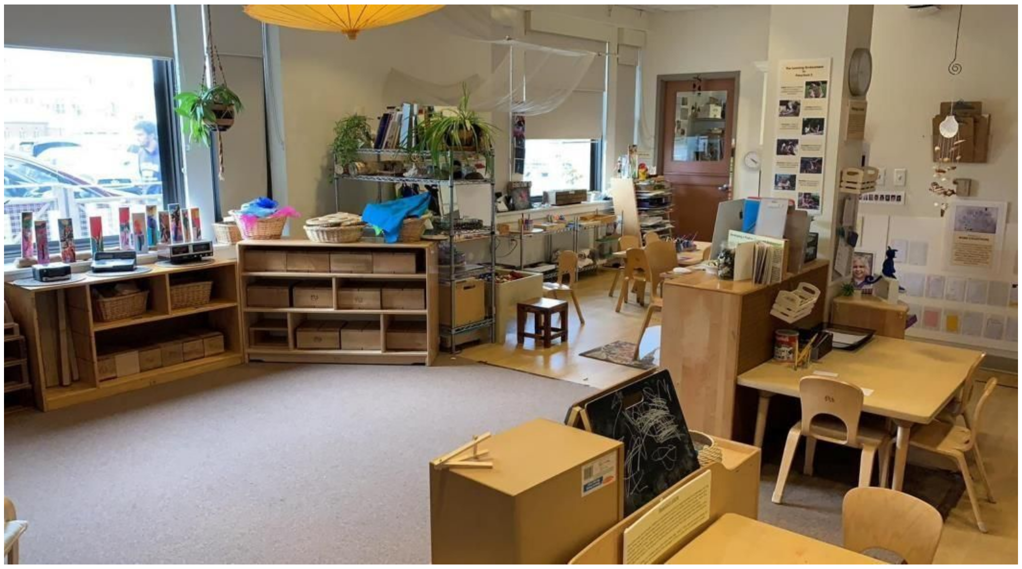


Figure 4: Clean room

3. System-level Requirements

The system-level requirements are divided into two categories: mandatory requirements and desirable requirements. Under each category, requirements are further classified as performance requirements that are functional requirements with qualitative measures, and non-functional requirements, based on their essence. The requirements originate from the project goal, derived from the use case, and validated through preliminary calculations and stakeholders' feedback.

There have been no changes in the mandatory requirements since the PDR since we believe we are on track to achieve all the requirements. We might modify M.P.9. depending upon future design iterations. In the desirable performance, the auto-charge requirement has been removed since our base does not have that capability unlike iRobot's Create 2, the robot base we started with.

3.1 Mandatory Performance Requirements

<u>The system will:</u>

M.P.1. Explore, scan and create a 2D map for 90% of the reachable area in a room.

M.P.2. Clean up a 20m^2 room with a dozen tennis-ball-sized objects within 30 minutes.

M.P.3. Navigate to a designated reachable location in a room with pose error < 10%.

M.P.4. Go over carpets and rugs with thickness less than 12mm.

M.P.5. Detect and avoid 95% of the obstacles with a clearing distance of 20cm.

M.P.6. Classify all tennis ball-sized objects with classification error < 20%.

M.P.7. Pick up and collect each classified object within 5 attempts.

M.P.8. Pick up at least 80% of the classified objects in the room.

M.P.9. Drop the clutter in a designated container with a success rate > 90%.

3.2. Mandatory Non-Functional Requirements

The system shall:

M.N.1. Operate autonomously.

M.N.2. Be mechanically safe (i.e. no sharp edges).

3.3. Desirable Performance Requirements

The system will:

D.P.1. Continuously operate for at least 2 hours once fully charged.

D.P.2. Have a sensing range of 15 cm to 4 m.

D.P.2. Have a physical dimension limit of 0.5 x 0.5 x 0.5 m.

D.P.3. Be affordable with a maximum cost of $5000 USD.

3.4. Desirable Non-Functional Requirements

The system shall:

D.N.1. Be easy to use by pressing buttons or through a GUI.

D.N.2. Have an inconspicuous, seamless appearance.

D.N.3. Be reliable and not get stuck or malfunction frequently.

4. Functional Architecture

The functional architecture aligns heavily with our use case and is color-coded with the Cyberphysical Architecture in a section below. As can be seen in the functional architecture, after CuBi is turned on, it is constantly looping through four major functions: exploring, categorizing objects, picking up objects, and dropping them off at a predefined location. It will continue repeating these major tasks until the room has successfully been decluttered. At this point, it will return back to its initial location and start re-charging.

One thing to note about the architecture is that the boxes marked with a red arrow require localizing, trajectory planning and moving. The boxes shaped like diamonds show functions that output decisions that determine what the robot will do next. Another thing to note is that Cubi will always start at the base.

The input for Cubi will be a user input and a cluttered room. In the current scenario, the input will be running a command from the terminal, but in the future, it should be a button press. The output of using our robot would be a clean room with a light indicating that Cubi is returning home.

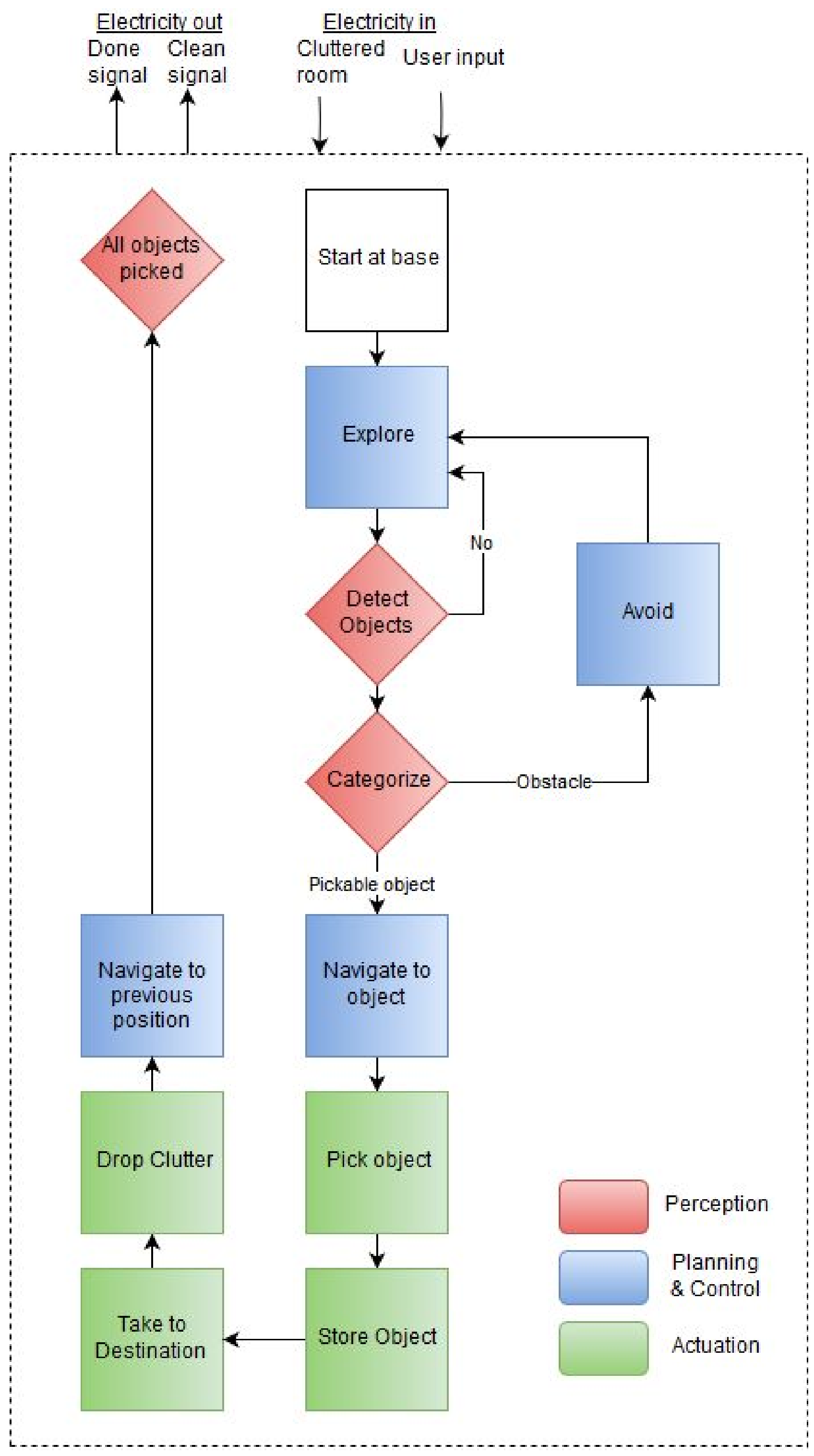


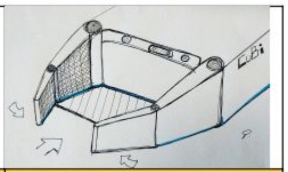
Figure 5: Functional Architecture

5. System-level trade studies

Value Ratings *	**Specs**	Person	Person using Manual Tool	Semi-Autonomous Robot	Fully Autonomous Robot
1: Inadequate	**Price (USD)**	N/A	$20	$2,000 - $20,000	$2,000 - $20,000
2: Tolerable	**Cost of Operation**	$20/hour to hire someone	$20/hour to hire someone	$10/hr	$1/hr
3: Adequate					
4: Good					
5: Excellent					
Criteria	**Weight Factor (100%)**	**Value (1 - 5) ***			
Save Time & Effort	20	1	2	3	5
Operates Autonomously	10	1	1	4	5
Operates Efficiently	5	1	2	4	5
Easy to Use	10	1	2	4	5
Seamless	10	5	4	4	2
Safe	10	4	4	3	3
Reliable	10	5	5	3	3
High Coverage Areas	10	2	3	4	5
Affordable	15	5	5	3	2
Weighted Average	5	**2.8**	**3.15**	**3.45**	**3.85**

*** Subjective Value Method**

Trade Studies **Manipulator & Gripper**

	Concept			
	Specs	Industrial Robotic Arm	Hobby Robotic Arm	Domain Specific Manipulator
	Description	UR-3 Collaborative table-top robot	uArm Swift Pro	CuBi Custom (estimate specs)
	DOF	6 DOF	4 DOF	3 DOF + 2 under actuated
	Repeatability	Repeatability +/-0.1mm	Rep +/- 0.2mm	Rep +/- 0.2mm
Value Ratings *	**Footprint**	Dia 128mm	150 x 140mm	150mm x 5mm each
1: Inadequate	**Weight**	11.2Kg	2.2Kg	1.5Kg including motors
2: Tolerable	**Payload**	3Kg	500g	1Kg
3: Adequate	**Max Speed**	1m/s	100mm/s	100mm/s
4: Good	**Reach**	500mm	320mm	150mm
5: Excellent	**Cost**	$23,000	$800	$1,000
Criteria	**WeightFactor (100%)**	**Value (1 - 5) ***		
Speed	5	5	4	3
Reach	5	5	3	3
Payload	5	5	3	4
Affordability	10	1	5	5
Safety	20	5	4	5
Compact Size	15	3	3	5
Dexterity	5	5	4	3
Accuracy	5	5	3	3
Development Time	5	5	5	3
Seamless Design	5	1	3	5
Weight	10	1	3	5
Durability	10	5	3	4
Weighted Average	5	**3.2**	**3.3**	**3.95**

*** Subjective Value Method**

Trade Studies	Sensors				
	Specs	Active Structured Stereo Camera (E.g. IR Projection)	Passive Stereo Camera	Laser Scanner (LiDAR) (2D) Single Beam	Laser Scanner (LiDAR) (3D) 16 Beam
	Reference Model	Orbbec Astra	Stereo Labs ZED	RP LiDAR A2	Velodyne Puck
	Output	RGB + RGBD + Point Cloud	RGB + RGBD + Point Cloud	Point Cloud	Point Cloud
	Computation Cost for depth	Medium	Very High (Require GPU)	Low	Low
	Low light performance	Medium	Poor	High	High
	Rich Visual Data for CV	Yes	Yes	No	No
	Range	0.6 – 5.5m	0.5 - 10 m	0.15 - 12 m	80 m
	Resolution	High and Dense	Very High and Dense	Low	High
Value Ratings *	**Footprint**	165 x 30 x 40 mm	175 x 30 x 33 mm	73 x 73 x 40 mm	103 x 72 x 72 mm
1: Inadequate	**Weight**	300 g	159 g	190g	830g
2: Tolerable	**FoV**	60° (H) x 49.5° (V) 73° (Diag)	90° (H) x 60° (V) x 110° (Diag)	360° (H)	360° (H), ± 15° (V)
3: Adequate	**Data Interface**	USB	USB	USB	USB
4: Good	**FPS / Scan Rate (LiDAR)**	30 @720p	60 @ 720p	Scan rate - 10Hz	Scan rate - 10Hz
5: Excellent	**Cost**	$150	$500	$320	$4,000
	Reflectance Issues	Adversely affected	Not greatly affected	Adversely affected	Adversely affected
Criteria	**Weight Factor (100%)**	**Value (1 - 5) ***			
Applications					
Object Detection	12	5	5	2	4
Object Identification	9	5	5	1	2
Localization	14	3	3	4	5
Performance Factors and Non-functional requirements					
Output information	12	5	5	2	3
Range	8	4	4	4	5
Affordability	8	5	4	4	1
Low light performance	7	4	3	5	5
Computation requirement	7	4	2	5	4
Reflectance issues	4	3	5	3	3
Resolution	5	4	5	2	5
FoV	5	3	4	3	5
Accuracy	5	4	3	3	4
Size	2	4	4	5	3
Weight	2	5	5	5	3
Weighted Average	5	**4.2**	**4.04**	**3.19**	**3.77**

*** Subjective Value Method**

6. Cyberphysical Architecture

The cyber-physical architecture shows the interactions between the hardware and software components of the system. It is divided according to the basic functionalities of a robot: Sensing, Perception, Planning, and Actuation.

Cyberphysical connects each function in our architecture to a physical and information aspect of our system. For example, the 'Avoid Obstacle' function requires CuBi to sense, identify, and plan in order to accomplish it. Thus, our architecture has been designed in such a way that the subfunctions of each function relate to individual components of our system.

Our system was built using ROS. Cubi senses the environment using an Intel RealSense Camera and a Hokuyo LIDAR. This data is then processed by the Nvidia Jetson TX2. The processed data is used in two processes: to decide where to move to and to decide how to grasp. It is through grasping that Cubi will affect and alter its environment.

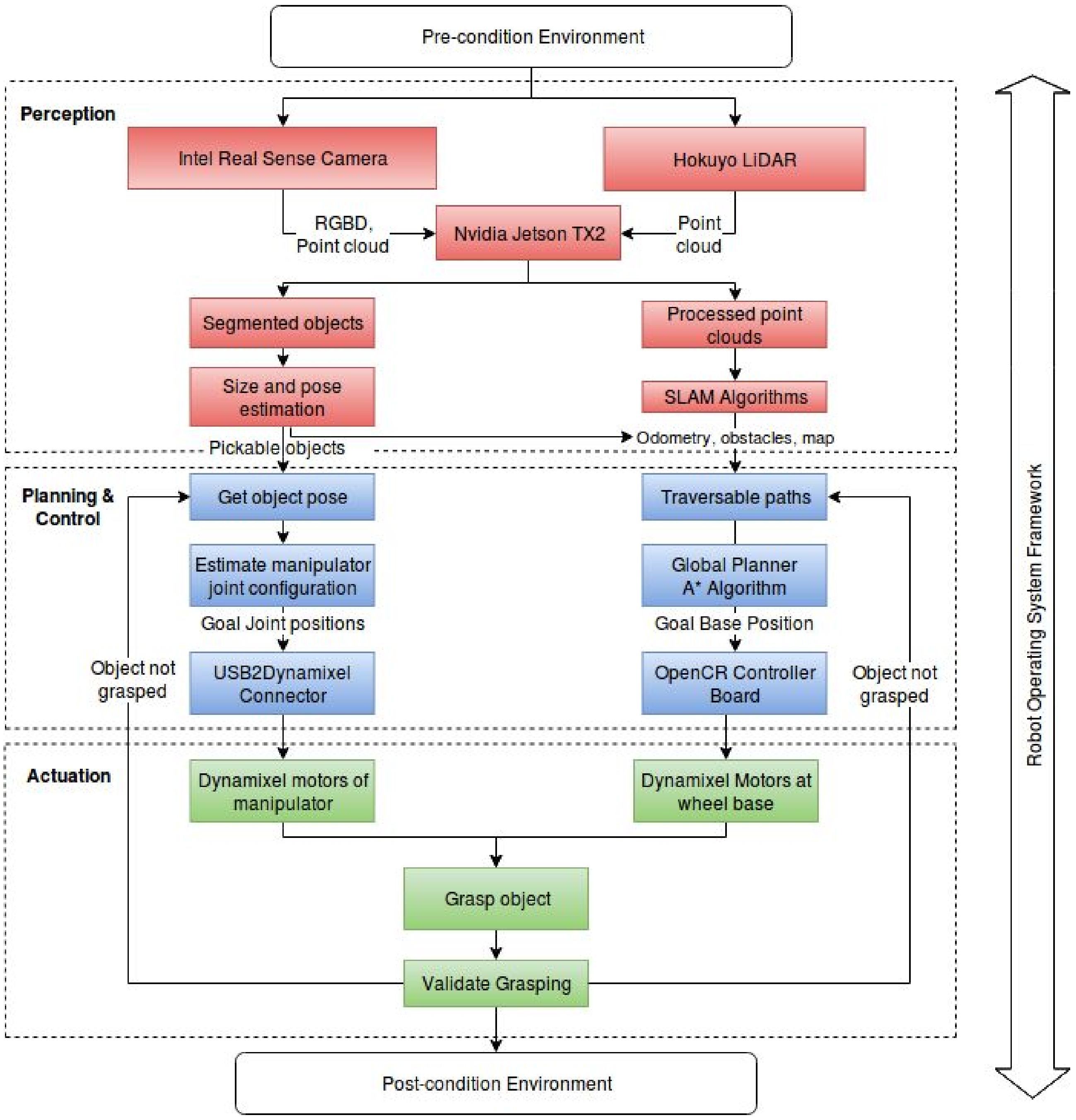


Figure 6: Cyberphysical Architecture

7. System Description and evaluation

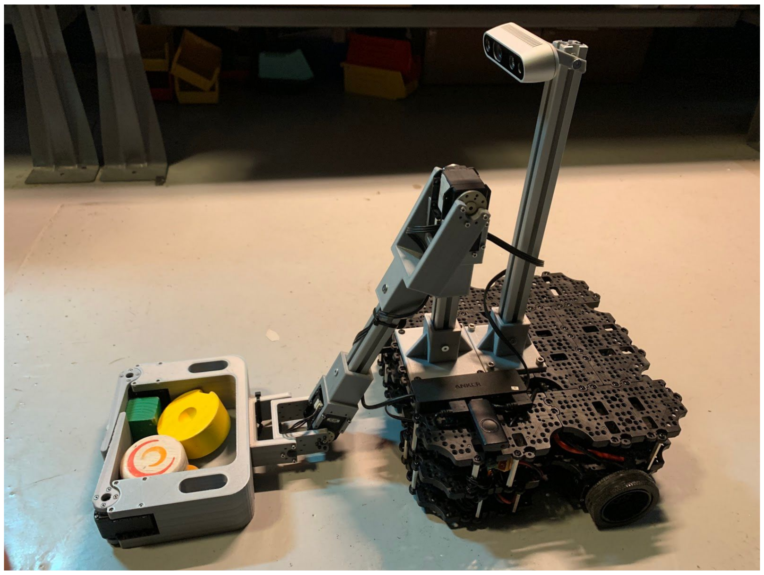


Figure 7: Cyberphysical Architecture

CuBi is an electro-mechanical stand-alone system, that integrates perception and actuation powered by an intelligent software which enables it to act on its environment autonomously. Figure 7 shows the completed assembly of CuBi that we used for our demonstration.

The robot uses a Turtlebot 3 Waffle Pi[4] as its mobile base. The base has a manipulator fitted on the top layer and also has Intel RealSense D435[5] and Hokuyo lidar [6] mounted on it. It has a PCB for power distribution and an OpenCR [7] controller that is used for the mobility of the robot. The system is complete and functional and we have improved the design and functionality for robustness and performance.

7.1. System/Subsystems Descriptions and Depictions

7.1.1. Finite State Machine

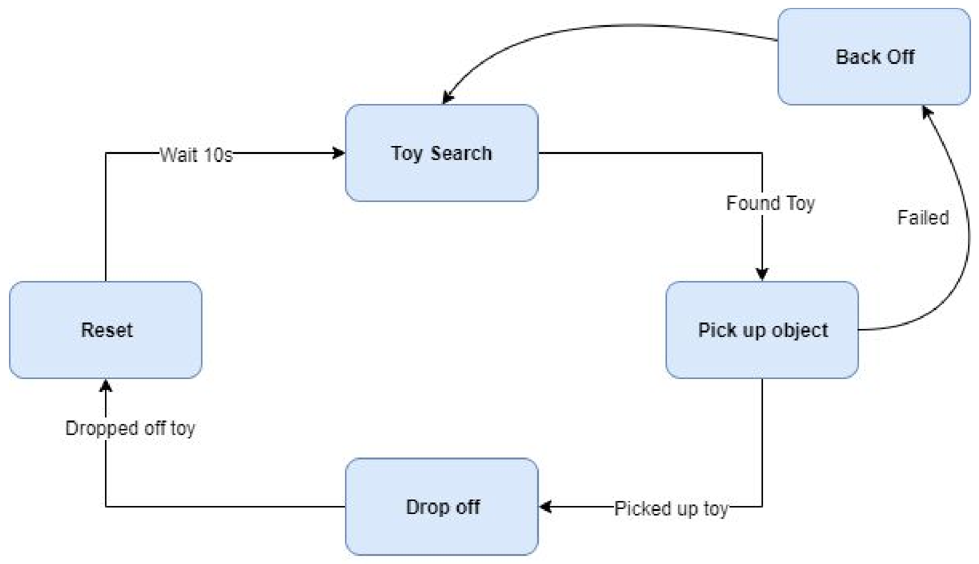


Figure 8: State Machine

The state machine as shown in Figure 8 is used by CuBi to perform its entire operation. When CuBi is turned on it goes on to ‘search mode’. In this mode, Cubi navigates and explores a room. Once an object is detected, CuBi switches to ‘pick-up object mode’ when it moves to the goal pose, picks up the object, and validates whether the object was picked. Then in the ‘drop-off mode’ it goes to the home position to drop off the object it picked up. In this mode, it goes to a point where it can go in a straight line to the home position without colliding into obstacles. Then it resets to the home position and starts searching for objects again.

We updated our State Machine by creating a new state called “Back Off”. We modified the code to switch states when the callback function above indicated that an object was trapped. The paddles are instructed to open, and the robot is in a fail-safe mode. In the State Machine loop, We added a routine that if the machine was in the “Back Off” state, to call a function that backs the robot off 10cm, and then switches states back to “Toy Search” state. For now, this is a simple heuristic that gives the robot a chance to try to pick the object again, possibly from a different angle.

7.1.2 Mobile Base

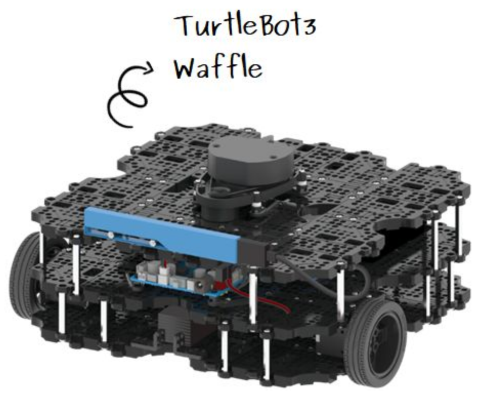


Figure 9: Turtlebot robot base

We have performed trade studies on various available off-the-shelf mobile robot platforms, as well as the option to build our own mobile base from scratch. Then we chose the TurtleBot 3 Waffle Pi[4] as our mobile base platform which is shown in Figure 9. The main reason to select this platform is the high-modularity design and capability to be easily modified and integrated with other components.

The base is powered by two Dynamixel XM430 motors with a differential drive steering mechanism with two caster wheels on the back. Dual large capacity Li-Po batteries are mounted on the base to power the system, ensuring at least 1-hour running time once fully charged. In the middle layer, an OpenCR microcontroller board is installed to control the chassis motors.

In order to improve odometry data precision, we successfully designed, 3D printed, and installed a new, much more robust, rear caster for CuBi to better navigate on the carpet with its reduced drag. It reduced rolling resistance to the mobile base noticeably and as a consequence linear drift of our measurements.

7.1.3. Onboard computer, PCB & Power Management

The mobile base has an embedded IMU onboard, and continuously sends fused IMU and wheel odometry information to the onboard computer through a serial port. On the top layer, we have mounted the manipulator, the NVIDIA Jetson TX2 onboard computer, and the power distribution PCB that we designed.

Both software and hardware components of all other subsystems have been fully integrated together with the mobile base. The base can be controlled either by using a remote joystick or by receiving velocity commands sent from the mobility controller node through ROS.

To safeguard all the onboard electronic components, we have designed and built a power management PCB board. It supports dual battery input to enable a longer duration of the system, and six output ports that can deliver protected power to the onboard computer, sensors, and actuators. We have also integrated several protection functionalities into this PCB, including the over-voltage protection, over-current protection, reverse-voltage protection, and battery voltage monitoring. Everything has been shown in Figure 10.

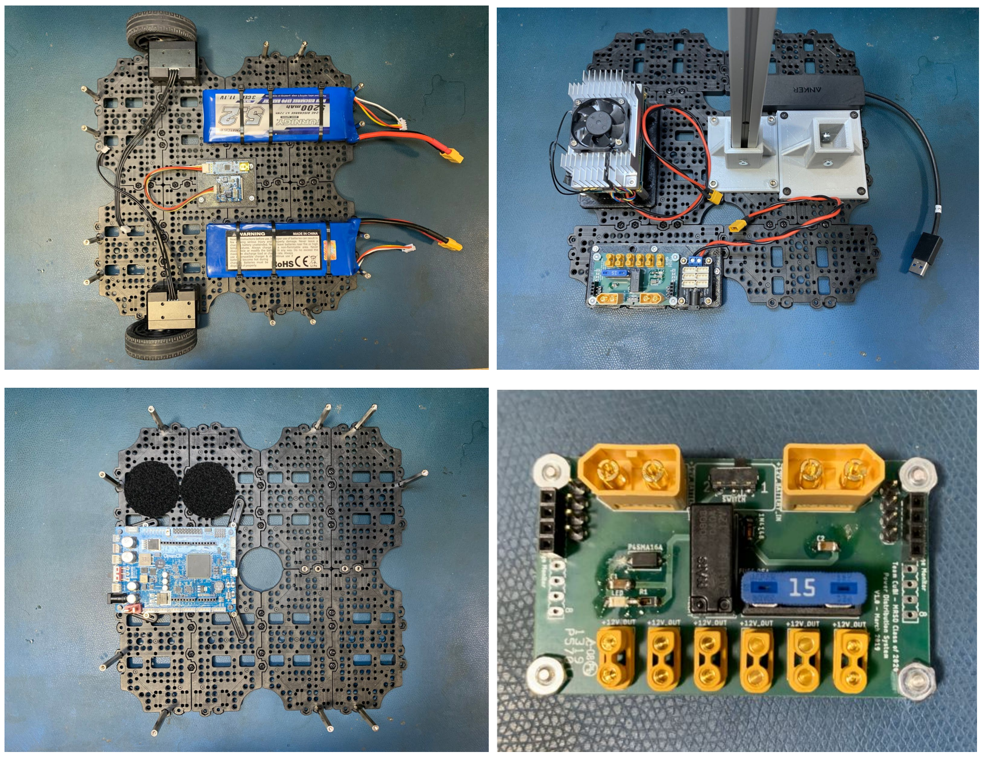


Figure 10: Tear-down of the Mobile Base and the Power Management PCB

7.1.4. Manipulator & Grasping

The manipulation subsystem is a crucial aspect of our project since our goal is to create a task-specific robot. This led us to design and fabricate a manipulator arm from scratch rather than using an off-the-shelf manipulator. We decided to use the caging mechanism for gripping as it generalizes grasping for a wide range of objects. Caging is a grasping strategy where an object's mobility is restricted using the end effector. The below figure depicts the design of the manipulator.

The manipulator as shown in Figure 11 is actuated by a pair of Dynamixel AX-12A motors for the gripper and a pair of Dynamixel MX-106 motors for the arm, making it a 4-DOF manipulation system. Each paddle is half the size of the tray which minimizes the possibility of the object getting stuck. The links are made up of 80/20 extrusion aluminum bars, and the brackets and tray are 3D-printed. We calculated a maximum payload of 500g for this manipulator.

We also addressed the issue of reacting to objects being trapped between the paddles. By introducing that state in our State Machine as mentioned above. We used the Dynamixel motors feedback "load" data to detect when an object was trapped. Fig. 12. below shows the respective paddle feedback while an object is being trapped between paddles. The small-amplitude curves show the motor torques, expressed as normalized loads, for operations when the object was successfully brought into the tray. It then became clear that any spikes with absolute values over 0.5 were an indication that an object was trapped for both scenarios.

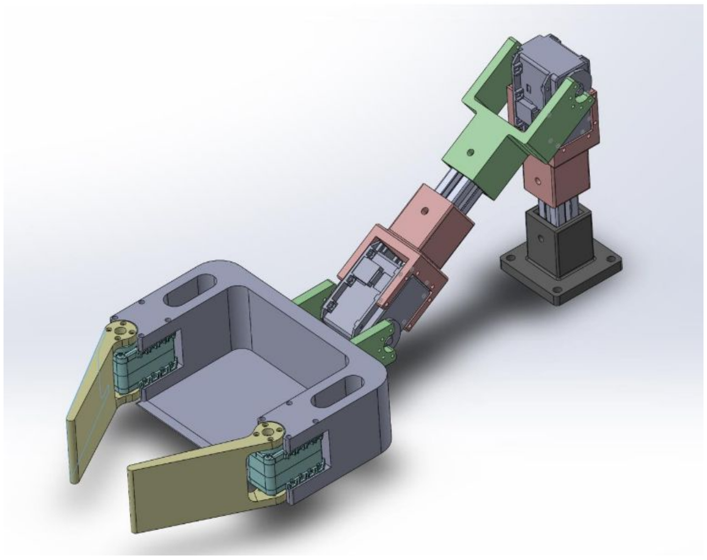
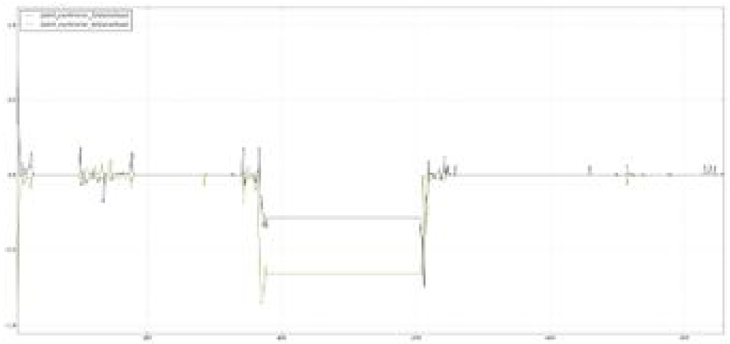


Figure 11: CuBi's manipulator



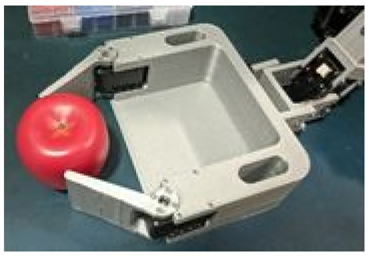


Figure 12: Torque analysis for fail-safe operation

7.1.5 Perception: Object Detection and Grasp Validation

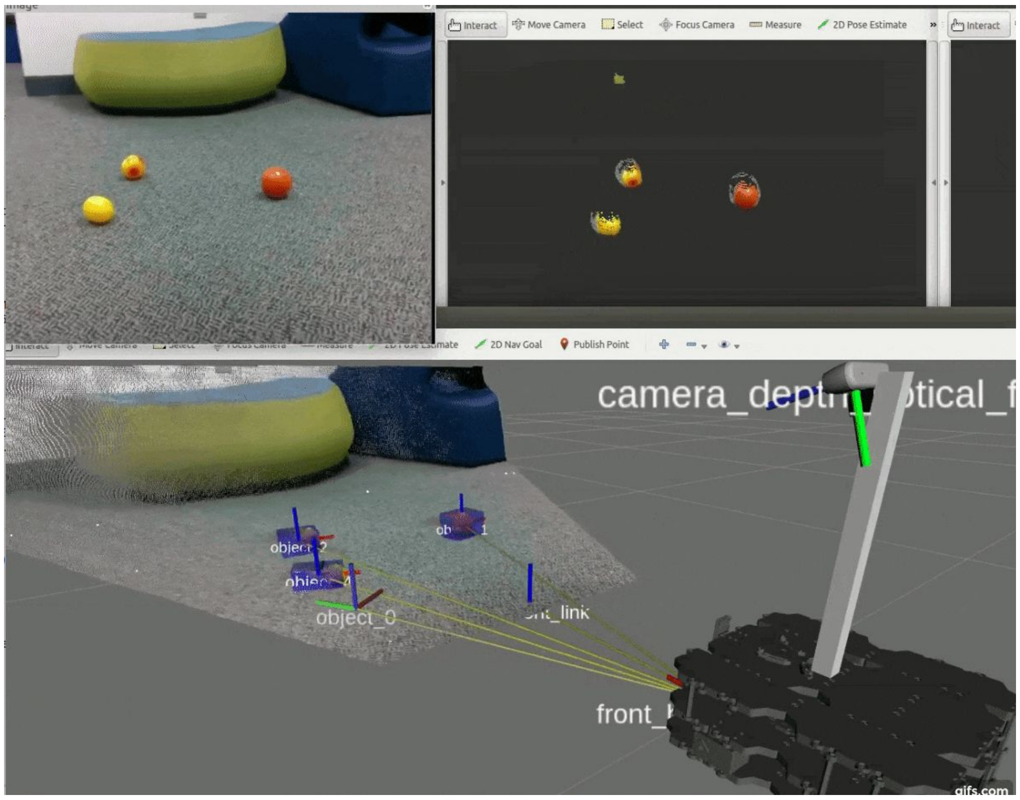


Figure 13: Perception visualization

Perception is used to sense the environment and to identify different objects. A stereo RGBD camera is used for obtaining rich visual RGB data for classification, as well as depth and point cloud for size and distance estimation. Combined information is used to estimate the size, pose, and type of object. A combination of techniques such as geometric vision, learning-based and probabilistic methods are used to classify objects into two categories: objects to pick and obstacles to avoid. Figure 13 shows how our perception pipeline works in real-time. Currently, classical vision techniques and PCL library is used to process the point cloud. A region of interest is cropped, noisy points are removed, RANSAC plane segmentation is performed, followed by clustering of remaining objects on the ground. A 3D bounding box is fitted around each detected object and the pose is estimated to classify them according to threshold criteria of position and size. Finally, object tracking is applied to uniquely identify these objects over time. To complement geometric features, learning-based methods with additional labeled or synthesized data will be used. This will also aid in obstacle detection. A list of detected objects with their absolute positions in the map (calculated with the help of relative positions with respect to the robot) will later be maintained and utilized by the planner for navigation.

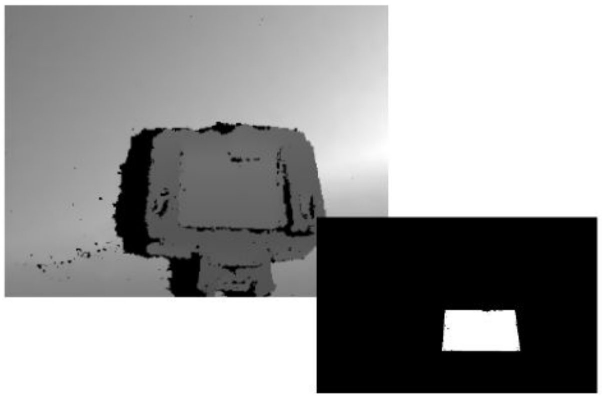


Figure 14a: Crop and store depth of empty tray

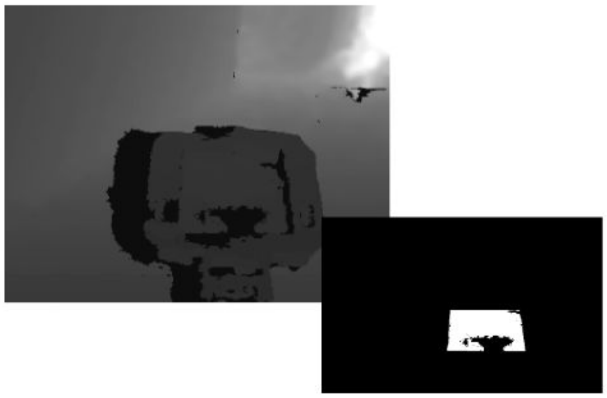


Figure 14b: Subtract image while validating

The following describes the pipeline for pickup validation. The manipulator is commanded to go to the fixed validation position on boot up. Then the depth image of the empty tray as shown in Figure 14a is collected and stored. After getting the pre-defined corners of the region to crop, this cropped portion is warped with inverse perspective mapping and rectified for better comparison as shown in Figure 14a. Then the current depth image is obtained when the comparison is required as shown in Figure 14b. The same procedure is followed as the empty tray. Now for comparison, the difference of the depth images of the cropped regions in both can be used against a threshold to validate the presence of an object in the tray.

7.1.6. Mapping & Localization

In order to operate robustly in the environment with a potentially unknown starting location and the ability to recover from odometry drifting, CuBi must-have essential information regarding the environment. With the Hokuyo URG-04LX-UG01 LiDAR installed, CuBi is able to map the environment beforehand, and use the map to localize itself afterward by matching the laser scans to the map.

The LiDAR gives us a continuous stream of range reads up to 6 meters, forbearing angles between -pi/2 to pi/2 for every 0.3-degree increment. With that data stream, we use Hector-SLAM[8] package to create a map of our environment. After successfully building the map, it will be saved to the hard disk of CuBi's onboard computer. We then use the Adaptive Monte Carlo Localization (AMCL)[9] package, which essentially uses a KLD-sampling approach with a particle filter, to track the pose of the robot with respect to the known map. In the particle filter, the odometry information of wheel encoders integrated with IMU is used as the propagated initial guess. The laser scan matching result is used to update the filter. As the particle filter runs, it keeps updating the relative transform of the odometry frame with respect to the map frame, which corrects the odometry drifts.

Our testing proves that CuBi is able to localize itself well in the testing area, with respect to the pre-built map. It is also able to recover and re-localize itself after getting stuck, where wheel odometry may have already drifted with a significant amount. The built map is shown in Figure 15.

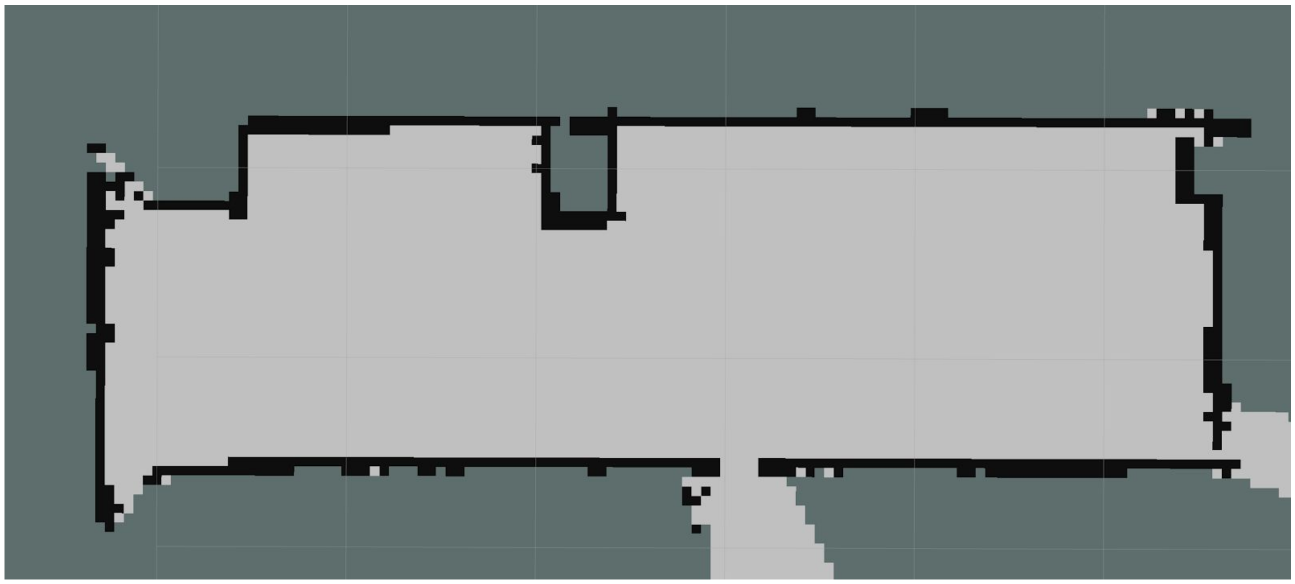


Figure 15: Map of the Testing Area

7.1.7. Planning & Control

The planning and control subsystem consists of an exploration module as the global planner, a local planner, and low-level controllers. The exploration module analyzes the map information and sends waypoints to the state machine, mentioned in Section 7.1.1. The details regarding the exploration module will be described in Section 7.1.8. The state machine then interacts with the local planner and controller. The mechanism is described as follows.

The local planner and low-level controllers consist of three layers. The outermost layer receives information from the state machine, the object detection module and the localization module. It utilizes the TF package to query any necessary frame transformations. Any target pose received will be transformed with respect to the map frame, and set as the goal. Once a goal is specified, the middle layer will then compute the relative error of CuBi's current pose relative to the goal pose. Then it generates a series of motion-defined below. Each motion is tied with a relative error in terms of distance or heading angle. This error is sent to the last layer which uses a position-based PID controller to compute and publish command velocities to the base motor controller board.

The middle layer of the local planner will generate CuBi's motion as follows::

- Rotate to orient directly towards the target position
- Move straight forward along the x-axis of the base frame to approach the target position
- Rotate until reaching the target orientation

In addition to the control of the base, the state machine also interacts with a node that controls the Dynamixel motors of the manipulator. The manipulator has a fixed set of configurations, such as moving, pick-up, drop-off, pick-up validation, box-approaching, etc. The entire pipeline is integrated using the Robot Operating System (ROS). The availability of ROS packages for Dynamixel motors and the Turtlebot base proved very beneficial to our cause.

7.1.8. Exploration

For exploration, the goal was to create a pipeline that was as flexible as possible. We wanted it to be robust to different room shapes and sizes. The major assumption we made was that a room has four main sides that define it.

The exploration module reads in the map and the metadata from the file. The occupancy grid image is very large and the room takes up approximately 5% of it. To speed up exploration analysis, we cropped it to the size of the room. We store how much was cropped, so we can map back the pixels into the original image. We then use RANSAC to find major lines created by the borders of a room. Once the lines were found, we clustered them into two categories using K-Means. We got the two largest lines in each of the groups to define the borders of the room. We later found the edges of the lines to create a convex hull. Finally, we set all points outside of this occupancy grid as obstacles. This border prevents Cubi from exiting the room through the small holes we have around the map. The process of refining and pre-processing the map is general and automatic.

Once the room was defined, we needed to decompose the cells. We restricted Cubi's range of movement between cells to be forward, backward, right and left. In this way, Cubi always enters the next cell perpendicular to the cell boundary and we can assure that it can scan the entire cell when going to its centroid. We decided the cell size should be 0.7m x 0.7m as Cubi can reliably see toys within a radius of 0.7m. This allows Cubi to scan one cell and parts of the ones in front and to the sides. This extra redundancy was created to ensure we pass through most sections twice.

To create the cells, we decided to use a conservative approach. The resolution of the cells in the stored occupancy grid was 5 cm. We grouped the cells, so they were decomposed into the size we wanted: 0.7m x 0.7m. When grouping the cells, if any of the smaller 5cm cells contained an obstacle, we would label the entire decomposed cell as an obstacle. This conservative approach acted like a way to inflate obstacles. Once we created the decomposed cell grid, we used the centers of all the cells as the waypoint.

As of now, the exploration policy is a very simple one that right followed the walls only visiting unvisited cells. Remember Cubi can only move in four directions. If it was surrounded by obstacles or visited cells, it would go to the closest unvisited cell. The ending condition was having visited all the waypoints. The exploration policy also takes into account the direction Cubi is facing, so that not only is Cubi sent to the right centroid but is also pointing correctly towards the next place it needs to go. Overall, this process allows Cubi to avoid obstacles found on the map and ensure that it has covered the entire room.

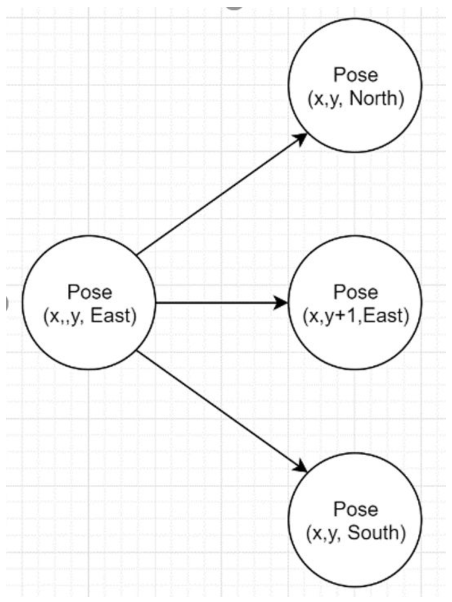
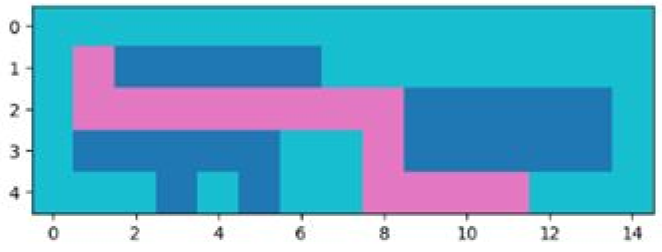


Figure 16: Cost graph representation

Finally, we also optimized our planner algorithm for situations when CuBi had to travel from point-to-point by reducing the number of turns it would take to complete the task. By using the graph representation shown in Figure 16.

We were able to succinctly add cost for state transitions between cells where a turn would be necessary. As shown below in Figure 17, CuBi traverses the same path, but the latter one, turning twice instead of three times. In practice that saves CuBi valuable time when exploring and collecting toys.



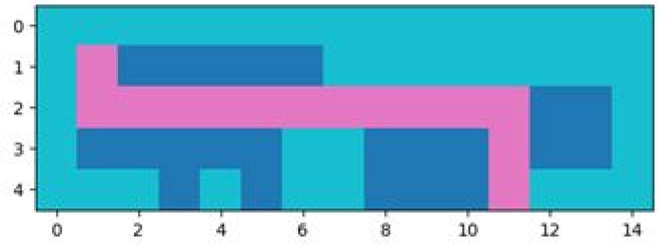


Figure 17: The second path shows an optimized route in which Cubi only needs to turn twice.

7.2. Modeling, Analysis, and Testing

As a proper system engineering design process, we have been doing a lot of modeling, analysis, and testing while designing and implementing our system.

Specifically, for the perception subsystem, we modeled the objects to pick up based on size, which is specified in our system functional requirements. We then implemented and fine-tuned our computer vision algorithm to classify the objects accordingly, with outlier rejection.

For the mobile base subsystem, first, we validated our maximum traction that the chassis motors and wheels can provide by testing the robot with the payload on different ground surfaces, such as smooth indoor floor surfaces and carpets. We then adjusted the weight distribution of the system, in order to avoid slipping on smoother surfaces.

We also evaluate the accuracy of wheel odometry, by comparing the reported value to the actual distance measured by hand that the robot traveled, to make sure that the amount of odometry drift was within the range specified in our performance requirements.

For the grasping subsystem, we designed, tested and iterated multiple generations of the paddle design of various lengths, roughness, and shape. Half the size of the tray as the paddle length was proven to be the most robust for grasping. Moreover, we estimated the maximum payload required for each joint of the manipulator and chose motors based on the calculated torque required.

For the electronics system, we calculated the power consumption of all the onboard components. Given the 1 hour running time as one of the performance requirements of our system, we then calculated the battery capacity needed and installed dual batteries based on that.

7.2.1 Test 1: Toy pick-up

Location: NSH B512
Equipment: CuBi and plastic toys
Subsystems: Perception (add visual odometry), Planning (reset odometry)
Metric: PR7 Odometry drift less than 10 cm.

Place five toys on the ground half a meter apart and half a meter in front of cubi. Turn cubi on. Have cubi pickup all toys.

7.2.2. Test 2: Toy delivery back to base

Location: NSH B512
Equipment: CuBi, plastic toys, and cardboard box
Subsystems: Storage Mechanism, Gripper Mechanics
Metric: PR8 Cubi can drop off toys.

Place two toys on the ground half a meter apart and half a meter in front of cubi. Turn cubi on. Have cubi pickup both toys bring them to the desired location. Repeat three times.

7.2.3. Test 3: Odd object gripper failsafe

Location: NSH B512
Equipment: CuBi and plastic toys
Subsystems: Gripper Actuation (not allow for an overload of motors), Gripper Mechanics (change design if needed)
Metric: PR9 Cubi can pick up flat toys.

Place two flat toys on the ground half a meter apart and half a meter in front of cubi. Turn cubi on. Have cubi pickup both toys bring them to the desired location. Repeat three times.

7.2.4. Test 4: Local planner with obstacle avoidance (NOT PERFORMED)

Location: NSH B512
Equipment: CuBi and plastic toys
Subsystems: Local Planner (obstacle avoidance)
Metric: PR10 Does not crash into any obstacle

Have CuBi follow 10 waypoints and place obstacles (chairs, large toys, feet, etc…) between all of them.

7.2.5. Test 5: Global planner and mapping

Location: NSH B512
Equipment: CuBi and plastic toys
Subsystems: Global Planner, Perception (SLAM)
Metric: PR11 Intersection area between cubi's reachable map and ground truth is greater than 90%.

Manually create a map of the reachable space of a room. Allow cubi to create a map of the room for 15 minutes and compare both manual and automatically created maps.

7.2.6. Test 6: Pickup toys under time constraint

Location: NSH B512
Equipment: CuBi complete system and subsystems
Metric: PR12 Number of toys at the desired location at the end of 10 minutes.

The independent party will place 5 toys at predefined reachable areas and will turn cubi on. All the toys which end up in the designated location will be considered picking up.

7.3. FVD Performance Evaluation

The high-level goal for our Fall Validation Demonstration (FVD) is to pick up 4 out of 5 toys (Figure 18) in a room in under 10 minutes. It was able to pick up all five toys in only 9 minutes. Examples of the toys can be seen in the figure below.

Figure 18: Examples of toys in our test set.

With the extra time, we tested Cubi again. During the second test, we asked the evaluators to try to break our system by placing smaller toys, plastic cups, tape and tape measures. Our system would always detect the toys. However, 5% of the time we would miss them when approaching them. This was because our vision system is not robust when detecting toys in motion. In terms of picking up objects, we were able to pick up all objects over 95% of the time if we aligned properly. We had difficulty when trying to pick up a plastic cup as the fingers on the gripper were not long enough to push it in. In this case, our system was able to detect that it had not picked up an object and that the plastic cup was stuck in the fingers.

Team D FVD - CuBi

Objective: Validate all of CuBi's subsystems to meet all the performance requirements as described in Table 1.

Location: NSH B512. It will have obstacles like chairs placed in the area. The area will be closed with walls created by furniture. Carpets in the area will have a maximum thickness of 1.2 cm.

Equipment: 5 tennis ball-sized toys, CuBi, box, any necessary replacements for any major subsystems which are at high risk of breaking, AprilTag

Setup:

1. Before testing, Team D will have already created a 2D map of the walls of the room.
2. Third-party places 5 toys at least 30 cm away from obstacles (including walls) at any location they want.
3. Team CuBi will place a box with AprilTag at the starting location. This will designate where the toys will be placed by the end of the *10 minutes*. Toys will be placed in a box.
4. CuBi is placed in the designated starting position.

Procedure:

#	Description	Performance Measures
1	Launch CuBi.	
2	CuBi starts to explore the room and perform SLAM to add static obstacles to 2D map of the room.	90% of the reachable area should be mapped by the robot
3	CuBi performs local planning to traverse the room while avoiding the obstacles it sees.	Avoid 75% of the obstacles
4	Whenever CuBi sees a toy, it uses its manipulator to pick it up.	Manipulator should pick up toy within 5 attempts

5	CuBi should be able to drop the clutter at the designated position marked with AprilTag accurately.	The success rate of dropping should be more than 90%.
6	CuBi will reset its odometry every time it drops off a toy.	Will localize indoors with accumulated error < 10% per 20 minutes of operation.
7	CuBi should be able to pick up most of the toys off the ground.	At least 80% of the toys should be picked up by CuBi
8	CuBi should be able to clean up 20m² area in a reasonable time.	It should pick four out of five toys in less than 10 minutes

Table 1: FVD plan

Figure 19 : MRSD Fall Validation Demo Encore

7.4. Strong and Weak Points

Strong points

One of the aspects which makes the system robust is our accurate object detection and pose estimation algorithm. It is very good at filtering out what is not clutter and the number of false positives are very few. We shifted to a new testing location with very different lighting conditions and the pipeline could be adjusted to adapt to the new environment without hurting performance.

Cubi's caging strategy is very good at generalizing the objects we can pick up from the ground. Even though we had never tested with picking up small objects like markers, it was able to do so.

The fail-safe grasping and object pick up validation pipelines are also very robust and helps in some vital time during a demo by eliminating tough objects to pick and false positives.

We have written several modules from scratch like the exploration package, state machine, and controls which provided us a lot of flexibility to customize the code according to the changing need was very helpful when integration was required.

Open-source version control software GitHub and team management software Notion were extensively used this semester to increase collaboration and productivity. Project management was taken very seriously and we had an independent project manager to perform task tracking guide us through the semester.

Finally, Cubi has a very clean, and accessible assembly and wiring. This makes it very easy for us to charge the batteries or swap any necessary components. We have spent very little time debugging this aspect thanks to having all the components clearly labeled and visible.

Weak Points

One of the biggest aspects that need to be improved is CuBi's crashing into walls when very close to the walls or big static obstacles. A check needs to be performed for the direction which has more free area.

In addition, CuBi's operating frequency is very low and can be increased for smoother operation.

Furthermore, Cubi is made of 3D printed parts and operates autonomously. This increases the risk of breaking parts and joints.

We were not able to integrate non-pickable object detection. We were, however, able to demo it independently.

8. Project Management

8.1. Team and skill-set

Figure 20 shows the skill set of the team members.

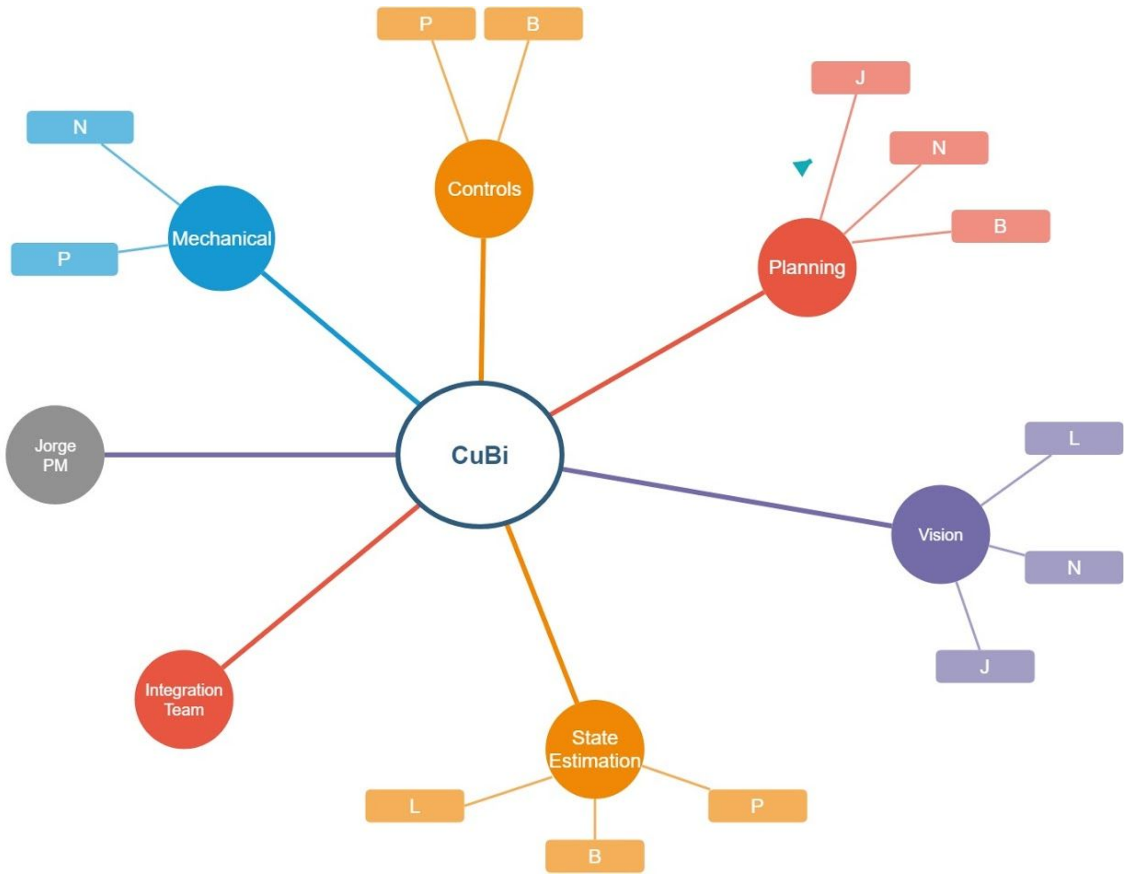
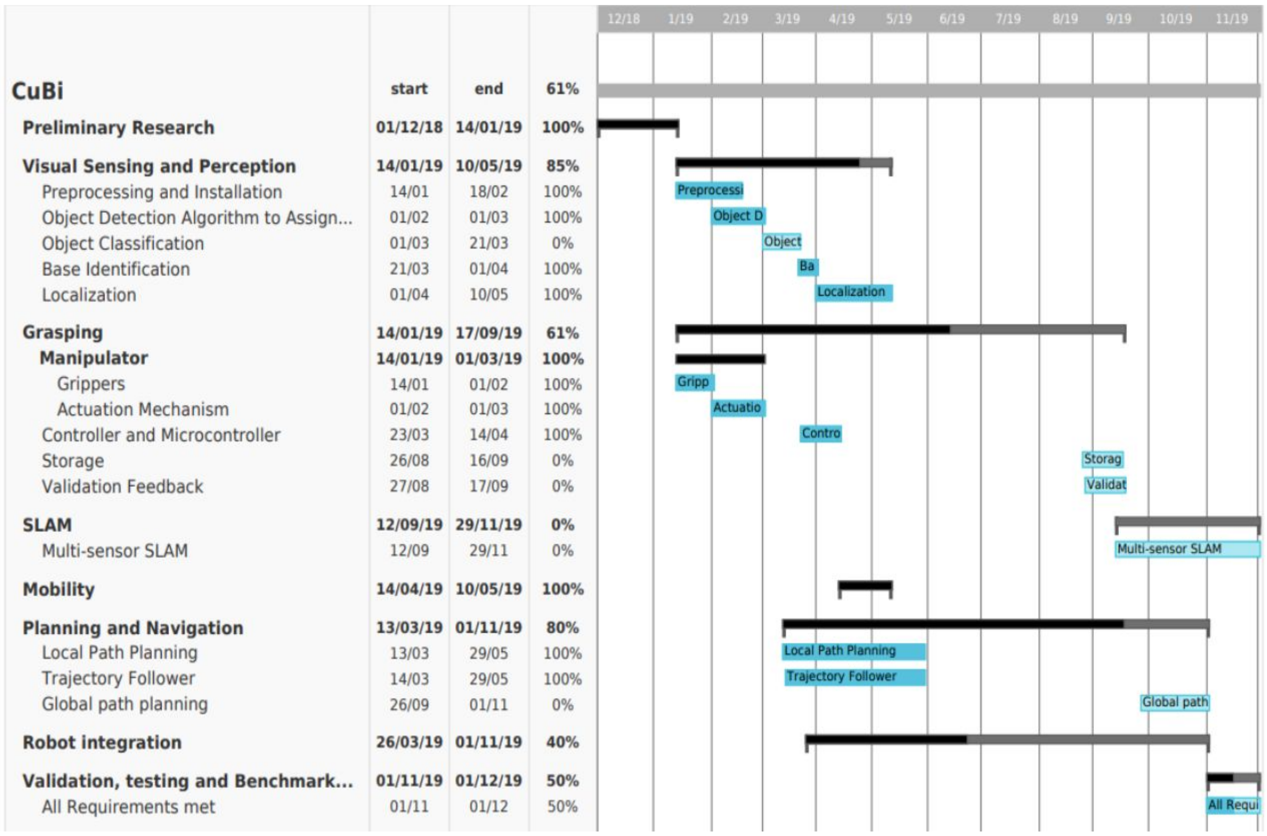


Figure 20: Team skill-set. B- Bobby, L- Laavanye, J- Jorge, P- Paulo, N- Nithin

8.2 Schedule



CuBi	start	end	61%
Preliminary Research	**01/12/18**	**14/01/19**	**100%**
Visual Sensing and Perception	**14/01/19**	**10/05/19**	**85%**
Preprocessing and Installation	14/01	18/02	100%
Object Detection Algorithm to Assign...	01/02	01/03	100%
Object Classification	01/03	21/03	0%
Base Identification	21/03	01/04	100%
Localization	01/04	10/05	100%
Grasping	**14/01/19**	**17/09/19**	**61%**
Manipulator	**14/01/19**	**01/03/19**	**100%**
Grippers	14/01	01/02	100%
Actuation Mechanism	01/02	01/03	100%
Controller and Microcontroller	23/03	14/04	100%
Storage	26/08	16/09	0%
Validation Feedback	27/08	17/09	0%
SLAM	**12/09/19**	**29/11/19**	**0%**
Multi-sensor SLAM	12/09	29/11	0%
Mobility	**14/04/19**	**10/05/19**	**100%**
Planning and Navigation	**13/03/19**	**01/11/19**	**80%**
Local Path Planning	13/03	29/05	100%
Trajectory Follower	14/03	29/05	100%
Global path planning	26/09	01/11	0%
Robot integration	**26/03/19**	**01/11/19**	**40%**
Validation, testing and Benchmark...	**01/11/19**	**01/12/19**	**50%**
All Requirements met	01/11	01/12	50%

Figure 21: Schedule

Summary

As seen in Figure 21, overall, we completed all but one requirement that we promised in January 2019. We definitely jumped in our ability to meet deadlines once we started to use Notion as our project management tool and our Fall semester was a success in terms of what we achieved. Jorge was also officially set as a project manager leading to much more rigorous planning of tasks, evaluation of objectives, and the ability to keep people accountable. Still, we faced three major problems: unnecessary dependencies early on causing people to wait on others, initial testing needed to be more rigorous, considering a task "done" when it was written as opposed to when it was integrated, and not having everyone know how to run Cubi. As the semester progressed, we learned from the mistakes and ended up getting really good as a team to meet deliverables.

8.3 Budget

Component	Manufacturer	Part #	Unit Price	Quantity	Total
Mobile Base	TurtleBot	TurtleBot 3 Waffle Pi	1399	1	1399
RGBD Camera	Intel	RealSense D435i	199	1	199
Computing Platform	Nvidia	Jetson TX2	599	1(Inventory)	0
Laser Scanner	Hokuyo	URG-04LX-UG01	1115	1(Inventory)	0
Servo Motor	Dynamixel	MX-106T	499	1(Inventory)	0
Servo Motor	Dynamixel	MX-64T	299	1	299
Servo Motor	Dynamixel	MX-28T	219	2	438
USB Converter	Dynamixel	U2D2	50	2	100
Connector	Dynamixel	ROBOTIS FR12- H101K	33.9	2	67.8
Connector	Dynamixel	ROBOTIS FR12- S101K	23.9	2	47.8
Battery	Turnigy	LiPo Batteries, Chargers 5200mAh 3S 12C LiPo	210	3	630
T Slot Extrusion	Zyltech	EXT-20-REG-1000- 10X	7.99	10	79.9
Aluminum Connector	PZRT	2020 Series	25.99	1	25.99
Toys	MDPlay	Fruits,Vegetables 14 Pc	28.00	1	28.00
Peripherals	Multiple	Keyboard Mouse Set	31.00	2	62
Misc Electronics	Multiple	Misc	200	1	200
					3576.49

Table 2: Details about the budget and money spent

The total budget of the project is $5,000. As seen in Table 2, currently we have spent $3,576.49, which is 71.52% of the total budget. We have some big-ticket items, like the TurtleBot 3 Waffle mobile platform, Dynamixel servo motors and the onboard computer, which comprised most of our budget. Overall, we were good with budget management.

8.4 Risk Management

ID	Risk	Type	Description	L	C	Mitigation
1	Manipulator	Technical	The design of the manipulation system should be robust to pick objects from the ground.	5	3	Iteration and validation of manipulator design.
2	Robot Damage	Technical	Crash into walls may cause damage to robots	4	4	Order extra parts and print replacements.
3	Plane segmentation	Technical	Different surface textures for plane removal poses a problem	2	4	Combine RGB images with stereo point clouds.
4	Low-light conditions	Technical	Low-light or differing light conditions can affect vision algorithms	4	3	Sensor fusion can help mitigate.
5	Dynamixel motors	Technical	Control of inventory Dynamixel motors is proving hard	5	4	Troubleshoot with help or replace it with other motors.
6	Auto-reset	Technical	CuBi might get stuck during operation requiring a hard reset	2	3	Use Laptop to stop operation and re-start.
7	Few False positives	Technical	Due to lighting variations, there might be some false positives.	4	3	Perform locking of objects by aligning towards it with consecutive frames. Add pickup validation to save time.
8	Manipulator gets stuck while picking tricky objects	Technical	The difficult orientation of the object or wrong size estimation.	4	3	Perform failure recovery and re-attempt pipeline.

Table 3: Risk Management

Table 3 lists the important risks that we anticipate encountering in our project. Out of all the risks, we have majorly dealt with risk ID: 1,2,5, 7, 8. For risk ID 1 which is a manipulator might not be able to scoop objects, we were able to get good results for our test set objects. However, it cannot generalize for a large number of objects. But this is expected since grasping is still an unsolved problem. Risk ID 5 has also reduced in number since now we have the experience of troubleshooting Dynamixel motors. We also purchase additional motors as a backup in case if we can't fix it. Solving risk ID 6 helped us to save time while conducting various tests for the demo. Risk ID 7 and 8 have been added during the course of the project and the mitigation was demonstrated during the FVD. The implementations helped us save significant time. Risk Matrix is shown in Figure 22.

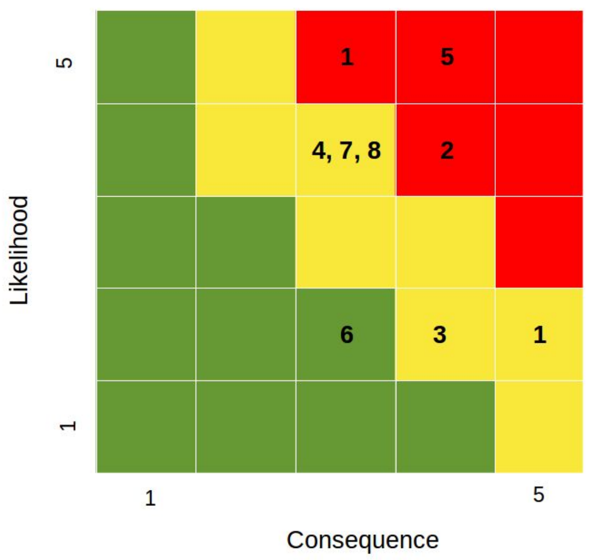


Figure 22: Risk Matrix

9. Conclusions

9. 1. Key Lessons Learned:

- This semester we decouple our tasks and member a bit too much. We understand that making individuals working on separate sub-systems independently, is an efficient way to get things. However, towards the end, we decided to work together again a team that led to better decisions and a more robust system. Our conclusion is that we should stagger individual work, with teamwork, and iterate until convergence of a system.
- We also changed completely the dynamics and rigor of our team. As each person or pair develops a subsystem, it is not considered done, until such sub-system is integrated into our pipeline. We found many times, that designing or coding a solution in isolation is typically easier than the task of integrating into the system. So we all became liable and responsible for development to integration.
- Utilizing GIT branches and discuss before pushing code onto the robot. No matter how

small the change, never push changes to the robot without testing. It is also important for us all to develop on separate branches so that when debugging, we know exactly what changes were made.

- Do not over-commit for each progress review. We usually send ideal case scenarios as our progress review goals, but we should be a bit more conservative.
- Reach a point where you can integrate everything as soon as possible to allow early and constant validation
- The BIG ONE: always freeze the code a few days before demo day. Never keep changing code, even if with good intentions. Robotics systems are exposed to many unforeseen variables, and so even seemingly innocuous changes might be enough to completely disrupt a system.

9. 2. Future Work:

The main thing we would need to improve on is improving the vision pipeline. It had a lot of false positives at the edges of the room and was not able to detect the pose of objects accurately when moving. Apart from this, we would also work on the following:

- Advanced global path planning: Create more optimized routes for exploring the room instead of just using the right following.
- Faster smooth trajectory following: We penalize speed to get a smooth movement of cubi. It would be interesting to see how quickly we can make cubi without intimidating feeling.
- Integrate dynamic collision avoidance: Perform collision avoidance in real-time.
- Manipulator design improvement: Modify the design of the manipulator to better generalize to objects. The main aspect we are trying to fix is the ability to pick-up flat objects.

10. References

[1] https://www.irobot.com/

[2] http://www.theoldrobots.com/Roboscooper.html

[3] https://www.cmu.edu/news/stories/archives/2016/may/robots-clutter.html

[4] http://emanual.robotis.com/docs/en/platform/turtlebot3/specifications/

[5] https://www.intelrealsense.com/depth-camera-d435/

[6] https://www.hokuyo-aut.jp/search/single.php?serial=166

[7] http://wiki.ros.org/opencr

[8] http://wiki.ros.org/hector_slam

[9] http://wiki.ros.org/amcl