University of Florida Robot for IEEE Hardware Competition

Tran, Ken
Member, IEEE
Gainesville, Florida

Langford, Matthew
Member, IEEE
Gainesville, Florida

***Abstract*—This paper explains the purpose, reasoning and implementation behind the mechanical and software design of the robot built by the UF IEEE Hardware Team. The paper will go over various mechanical systems of the robot. These systems include: chassis, end-effector, linear rail, and elevator. In addition, the reasoning and implementation of the particle filter will also be explained.**

I. Introduction

The competition was based on the unique challenges of the shipping industry. In particular, the challenges was modeled after the problems faced in port of Norfolk, Virginia, where the competition was held.

> *"The IEEE SoutheastCon 2016 hardware competition is designed with the intention of simulating modern port logistics and its related traffic. This IEEE Roads port provides a challenging game of robotic skill and logistics. Each team has to successfully detect shipping goods on a barge (three types of shipping container) which are strategically placed in a harbor field. Correct shipping goods then have to be picked up and transported to the correct shipping zone, and they will be further transported by the boat, by rail or by truck. Each team will have 5 minutes to complete the task."* [1]

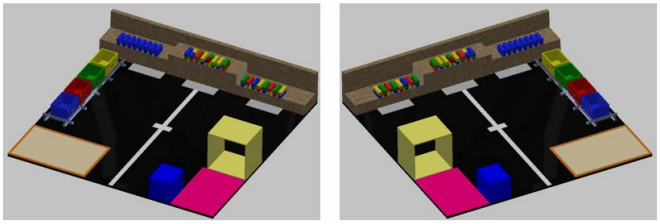

Figure 1: Overview of both field configurations. [1].

The robot has to fit within a 1 foot cube at the start of the match and expand up to a 20" cube once the match starts. Once the competition starts, the robot has to be autonomous for the duration of the match.

> *"Three zones with shipping containers at the barge are:*
> *A) Generic containers with no QR code, same size, one color (blue)*
> *B) Containers with QR tag, different colors (blue, red, green, yellow), 2 different sizes*
> *C) Color containers with QR code, different colors (blue, red, green, yellow), same size"*

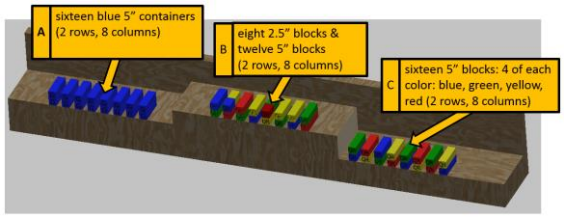

Figure 2: Different block zones. [1]

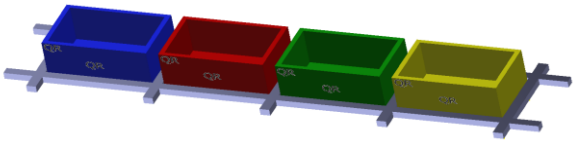

Figure 3: Rail shipping zone. [1]

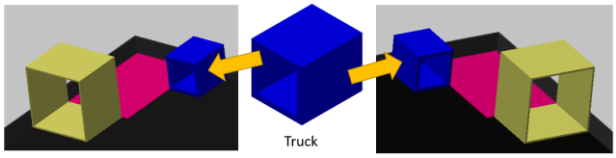

Figure 4: Truck shipping zone. [1]

II. Chassis Design

The chassis was designed to be as compact as possible while also accommodating for four Hokuyo LiDAR and a suspension system for the front axle. Everything in the chassis, except for the motor mount and wheel hubs, were manually machined to specifications. In addition, the chassis also carried an Intel NUC 64-bit computer and motor controllers for the driven wheels. The wheels were Mecanum wheels. These wheels allowed the robot to move in any direction regardless of its orientation. This attribute resulted is a great advantage in this competition, because it allowed the robot to move across in field in the XY plane while simultaneously being parallel to the walls with all the blocks. The robot doesn't have to waste time reorienting itself in order to move to the next challenge.

That being said, Mecanum wheels also came with a unique problem, all four wheels must contact the ground fully in order for the robot to strafe. This problem was brought on because of

the summation of torque allows the torque of some wheels to cancel each other out creating sideways motion. This summation will yield an undesirable net torque if one of the wheel don't provide the necessary torque. This problem was solved through a combination of a PID controller, which minimized wheel slips and a suspension system. The suspension system was essentially a seesaw with springs for resistances. As one wheel loses contact with the ground, the opposite wheel forces it back into place ensuring that both wheels are in contact with the ground.

Another important aspect of the chassis are the four LiDARs. The four LiDAR allows Shia to find his location on the course and navigate to the different tasks on the course. The LiDARs are mounted to the chassis through custom made L-Brackets. LiDARs are essential for navigation and it's one of the element that make Shia special. The use of four LiDARs opens up lots of options for the software team, the Particle Filter being one of those options, which allowed the team to implement a very robust solution that not only applies to this competition but many more competitions down the road.

Wherever Times is specified, Times Roman or Times New Roman may be used. If neither is available on your word processor, please use the font closest in appearance to Times. Avoid using bit-mapped fonts. True Type 1 or Open Type fonts are required. Please embed all fonts, in particular symbol fonts, as well, for math, etc.

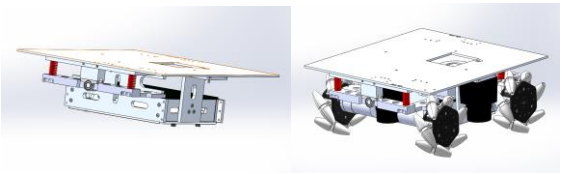

Figure 5: Chassis with suspension (left). Fully equipped chassis (right).

III. End-Effector Design

The end-effectors worked through a combination of computer vision, servos, and 3D printed parts. The camera looked at the QR codes and identified the color of the blocks. It also lines up the end-effector in the correct position and orientation. Upon activation, each servo moved the paddle that will hold the block in place. This way each servo "knew" the color of the block that it is currently holding onto. As the result, the servo can deactivate once the gripper is over the bin that corresponds to that color. There were two end-effectors on the robot. Each of these end-effectors can pick up four blocks for a combined total of eight blocks.

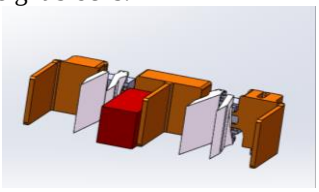

Figure 6: End-effector. Paddles and in white the orange is the main body. The red is a block similar t/o the ones used in the match.

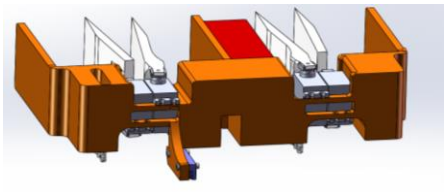

Figure 7: Embedded servos (in grey).

In order to accommodate for the large storage capacity, the end-effectors had to be as compact as possible. Servos were embedded into the main body to save space. In the contracted state, the end-effectors folded into each other to adhere to the size constraints.

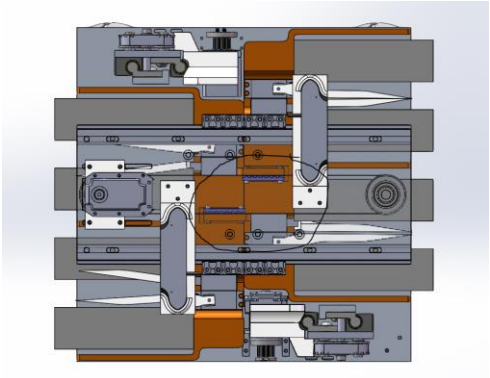

Figure 8: End-effectors folding into each other.

IV. Lateral Linear Rail Design

An opposite drive pulley system allowed the end-effectors to expand outwards and reach the blocks. The end-effectors are attached to a set of linear bearings that slides along the outer rails. The pulley system consisted of a Dynamixel 64-T servo at one end and a pulley at the other end both are joined by a timing belt. Each end-effector is attached to opposite side of a timing belts. This allowed the two end-effectors to move away from each other when the servo rotates one way and towards each other when the servo rotates the opposite way. This design allowed for the extension of the end-effectors to be controlled through the use of only one servo.

V. Elevator System Design

An elevator was necessary because the blocks sections are all at different heights. The end-effector must be manipulated. The solution came in the form of a multi-stage elevator system. There are 2 lifting mechanism (Fig. 9). These mechanisms allowed the linear rail system to translate along the Z-axis. Remember that the end-effectors are attached to the rails, so they move with it too. Each lifting mechanism is powered by a Dynamixel MX-64T Servo and a complex pulley system.

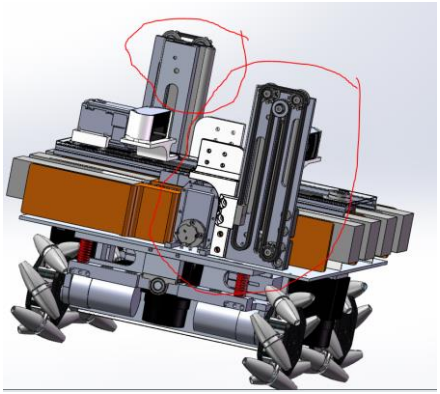

Figure 9: lifting mechanism are circled in red

VI. Particle Filter

Most teams who competed in the competition used basic forms of navigation that consisted mostly of 'dead-reckoning'. Our approach was much more complex since we had access to much more complex sensors - namely our four LiDARs. Our initial goal was to implement a simultaneous localization and mapping (SLAM) algorithm that would keep track of our exact location on the field at all times. However, an inherent problem with our model of LiDAR is that beams that reflect off of black walls would return data that was almost completely inaccurate and therefore we were not able to produce correct maps and our robot was not able to converge onto its position accurately enough for our goals since almost 75% of the walls of our field were black. The cause of this inherent error comes from the physics of how the LiDAR works and that black surfaces cause weird reflections of the beam that the sensor uses to estimate range. The solution to this problem was to rely on Bayesian filtering to estimate the state of the robot rather than relying on SLAM to pinpoint our actual location precisely. The state of the robot here refers to the robot's x position, y position, and its rotation relative to our predefined origin on the field.

The Bayesian filter implementation we used is called a particle filter. Our particle filter works on a basic principle: given an initial guess of the robot's starting state, the particle filter simulates a LiDAR scan that the robot would see assuming it were at that estimated location. This simulated scan is compared to the actual scan coming from the LiDAR and a weight is assigned to the position that the simulated scan was taken from - a weight closer to 0 means the simulated scan (and therefore the position that scan was simulated from) was inaccurate. Weights closer to 1 imply that the simulated scan was a more accurate representation of the robot's current sensor information (and also by extension that the initial guess was a good representation of the robot's state).

Now let's say that instead of testing one estimated state of the robot (one particle, if you will), the filter tests thousands of estimated states (thousands of particles) with slightly varied parameters against the actual LiDAR data coming from the sensors, the weights generated for each of these particles would produce a distribution of probability of the robot's current state - the maximum of which being the most likely state of the robot given the current data. After the top few percentile of particles are selected, the remaining particles are translated according to odometry measurements and new particles are generated around the average position of the particles. This process allows the particle filter to correct the position estimation that comes from the integrating the wheel velocity which tends to drift over time.

The particle filter has the benefit of not need perfect data since it will converge to the most likely position of the robot, not necessarily the actual position - which is the best you can hope for with the kind of LiDAR data we would get from black walls.

Simulating the LiDAR scans was the most computationally expensive part of the particle filter. Simulating a scan required the software to project a ray out from the sensors location and check for intersections with each wall of the pre-generated mathematical model of the map. If multiple intersections were found, the one with the shortest distance from the sensor were returned. Our implementation of the particle filter was unique in that instead of trying to do these computations on the CPU, we used OpenCL to simulate LiDAR data on the GPU. Since the GPU is a highly parallelized computation device, we were able to split the job up - instead of one processor handling the ray tracing, each GPU compute unit handled simulating one scan from a single particle. Offloading a majority of the navigation processing to the GPU allowed other mission critical processes to run and freed up computational resources.

Thanks to our implementation of the particle filter and an error correction algorithm we developed to run on the incoming LiDAR scans, we were able to converge to a very accurate estimation of the robot's state - within one centimeter of the actual location on the map resulting in superior navigation to any of our opponents.

VII. References

[1]https://docs.google.com/document/d/1ITIsL9fpTk5HKEJW1NENVrkgfCgXqONWpm0sevzeYmo/edit#heading=h.obp73v47rneu