

CS 189: Final Project Report

Ivan Cisneros, John Holland

Goal:

Our goal was to create a waiter robot that served drinks. The robot would wander the party offering guests the opportunity to place an order. After collecting orders, the turtlebot would return to the bar and submit the order list to the bartender, who would load up the turtlebot's tray with the proper drinks. It would then return to each of the guests who ordered delivering the proper drinks.

Implementation:

Mechanical Design

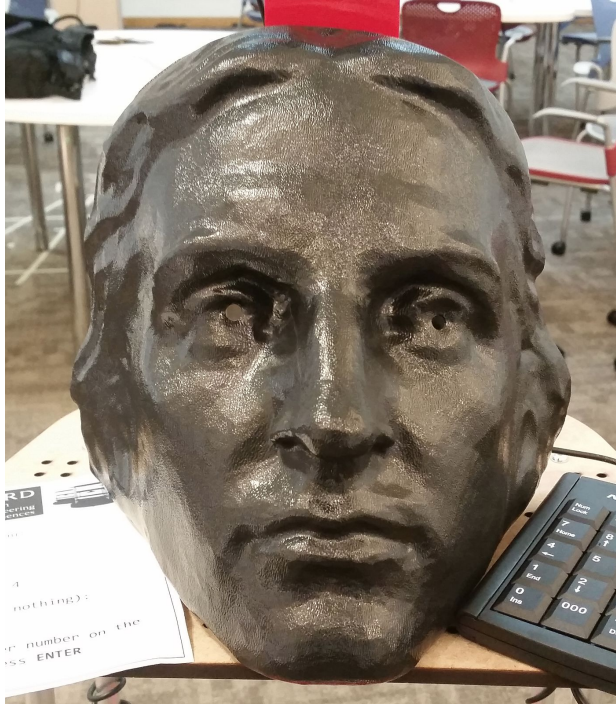
We made several mechanical alterations to our robot to make it more suited to its task. We laser-cut two acrylic plates to form a drink tray. Each has four small holes to allow it to slide onto the turtlebot's supporting rods. Each small hole has a set screw to keep the trays level and prevent them from sliding upside down. The bottom tray is flat to form a base for the drinks and the top has six large holes to hold a standard can.



We found part way through the project that an AR tag mounted at chest level rises above the camera frame when the robot approaches its target, so we laser cut and heat bent a stand to hold the camera higher up. It has a hole and an L-bracket to mount up to two cameras and two supporting beams to minimize shaking.

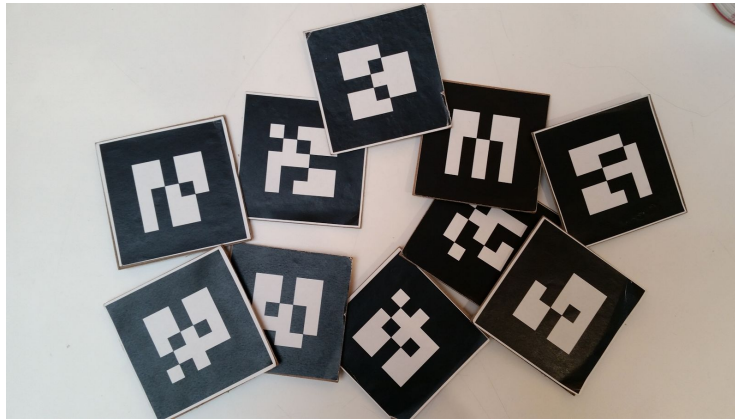


We also included a mask constructed using a 3D scan of John Harvard's head, a 3D printed mold, and a vacuum mold (initially constructed last October with help from the teaching labs). The speaker was placed behind the John Harvard mask in order to give the impression that the robot was actually talking.

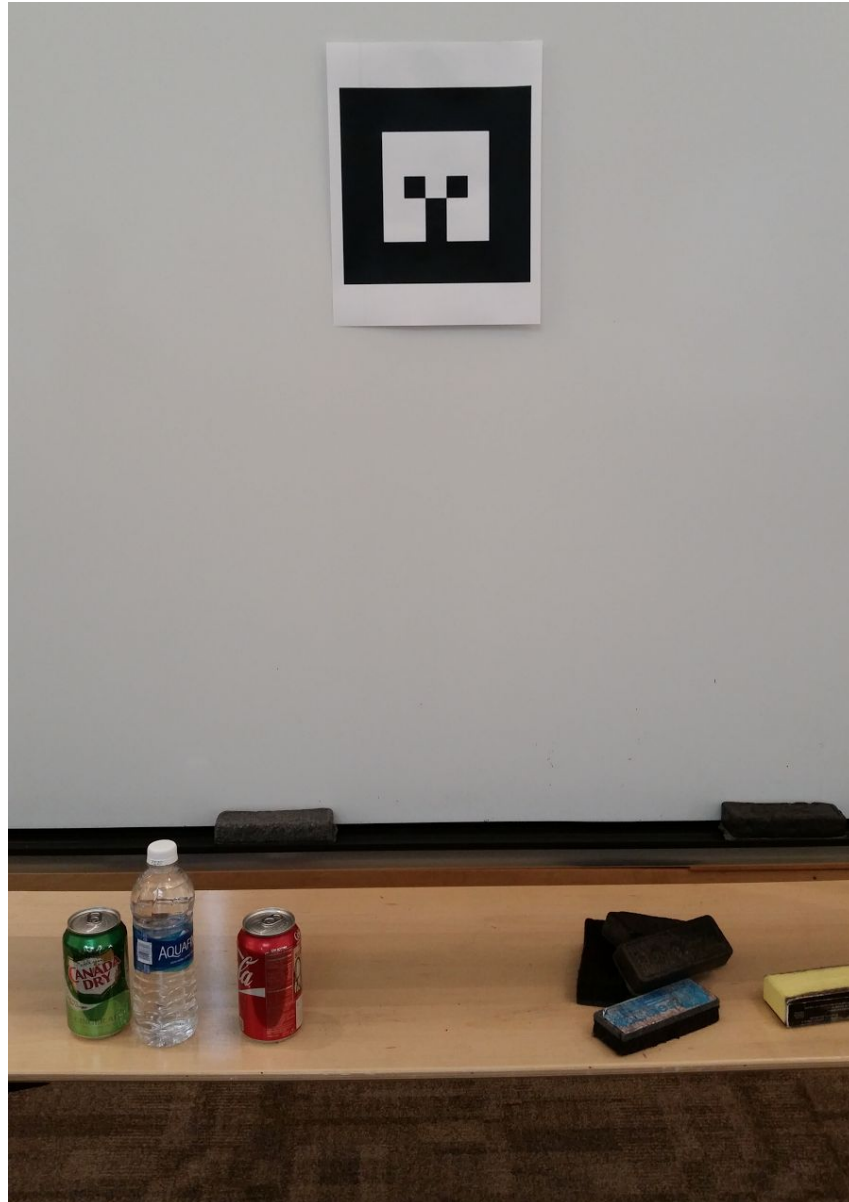


Processing AR Tags and Odometry Data

One of the project's biggest challenges was reliably using the AR tags. We chose to use AR tags as a means of homing in on people to take orders as well as homing in on the bar. They also provided a simple way of attaching an ID to each order so we could keep track of who ordered what and who had already ordered.



AR tags IDs that would be distributed to people at the party.



The AR tag that identified the bar area. This was predetermined to be the 0th tag.

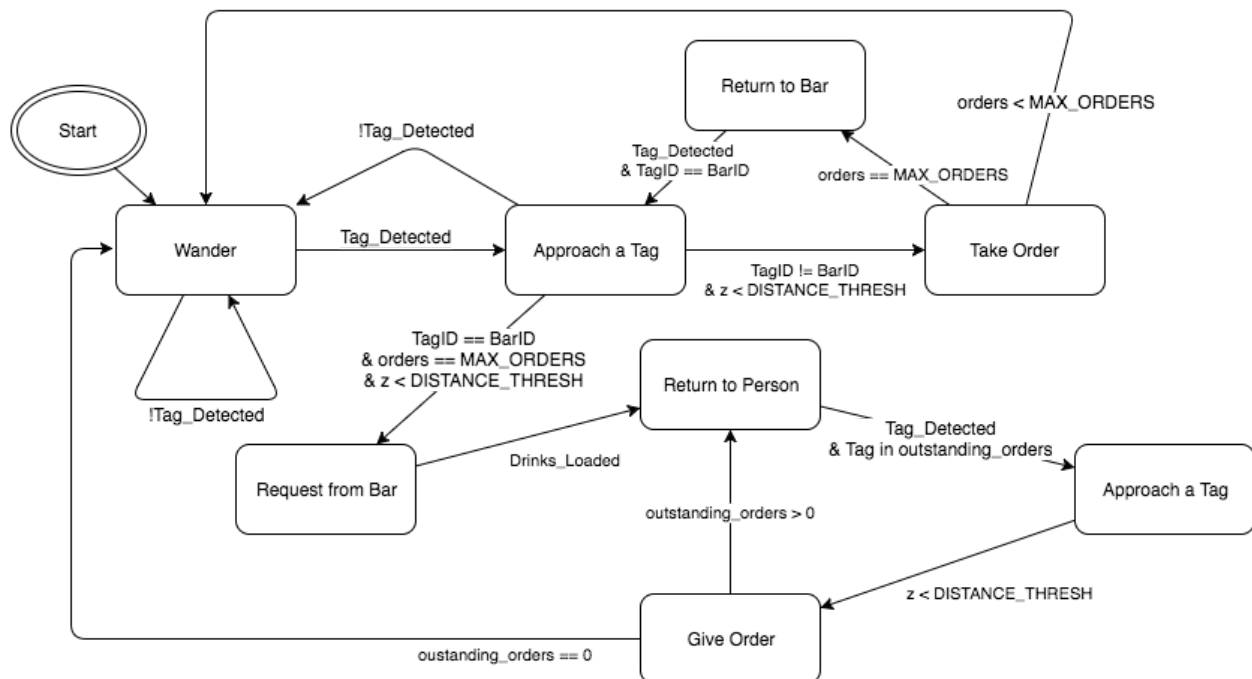
AR tags conveniently came with an ROS package, but documentation was poor so there was a lot of trial and error. One significant challenge was balancing processing speed with reliability. Using the camera in high definition allowed for very reliable detection of AR tags frame by frame, but caused large processing delays that made it nearly impossible to update move commands. We settled for a lower quality image, which allowed us to process more frames but made AR tag detection less reliable, often returning false negatives.

To compensate for the decrease in reliability, we created a 6-frame buffer. If the camera detected an AR tag in any frame over the course of 6, it would return that it had detected the

tag. The AR tag object was simple to work with because it provided the tag's location and depth, but the units were arbitrary so we had to determine thresholds experimentally.

We supplemented use of the AR tags with odometry data. Using the `robot_pose_ekf` package we were able to reliably keep track of the robot's position as well as the location of the bar and the location where each order was taken.

Finite State Machine Overview



We decided to implement functionality via a finite state machine because the waiting process required a fixed sequence of causal events (e.g. the robot should not be requesting order from the bar before it has gone out and taken an order from a customer). The above diagram shows an overview of our decision-making algorithm. Each state was its own function within the class. Though it was not possible due to the nature of an fsm, conflicting movement commands were handled by the fsm, which was the only function that could publish to the movement publisher.

Wandering:

Our wandering strategy was relatively straightforward. It was only used when the robot was taking orders and had not homed in on an AR Tag. The robot would randomly choose whether to move in a straight line or whether to turn slowly in a random direction to scan the room for Tags. Once it had chosen what to do it would choose a random time within reasonable bounds for which to perform this action. The wander function was implemented in such a way that it can easily be escaped if an obstacle or an AR Tag is detected. The limit to this strategy is that

there is no directionality, meaning that it would be ideal if the bar were at the center of the room, but in our circumstances this was not the case.

Obstacle Avoidance:

Obstacle avoidance mode took precedence over any other state where the turtlebot would be navigating, except when the turtlebot was in the final stages of its AR Tag approach. Obstacle avoidance consisted of two parts: bump detection and IR obstacle detection.

Reactions to bumps followed a fixed sequence of steps, where the robot would move in reverse for about 0.1 meters and turn to face a perpendicular direction from the side that was bumped. In this way, even if IR detection failed, the turtlebot would be able to navigate out of a corner relying solely on bump detection.

IR detection relied on a similar strategy to what we've used in previous assignments: a grid of "boxes" would span horizontally across a normalized distance image frame at a height slightly below the horizon. With every frame of IR data that was processed, the mean distance of each of the boxes would be calculated and compared to a threshold distance (0.7 meters). If the mean distance of the box was below the threshold, the box would be tagged as "triggered" meaning that there was likely an object within that area. If enough adjacent boxes were tagged as triggered, there was likely a big enough obstacle in that area of the frame (and it wasn't just a false positive). When enough adjacent boxes were tagged as triggered, the turtlebot would rotate away until this flag was deactivated.



Figure 1: An example frame from our IR camera, with the iterable boxes spanning horizontally (and vertically, in this case). The darker objects are closer than the lighter objects.

Returning to a Known Location:

Navigating to a precise location was difficult when relying solely on AR tags. Because of resolution constraints as well as the size of our badge tags, the webcam would often not recognize the AR tags at a distance greater than a meter. Thus, in addition to relying on the AR tags for hyper-local navigation, we also relied on odometric coordinates in order to give the turtlebot a more global sense of the locations of important places. Whenever the robot would be navigating to a specific location, it would first navigate to the coordinates of that location, then when it got within a certain distance of the coordinate, the robot would rely on homing in on the AR tag.

The turtlebot stored the locations of the bar and of every customer that placed an order. At initiation, the turtlebot would need to start within the bar's proximity. This is because the turtlebot would use the bar as a homebase of sorts, and would store its location as the origin in its internal odometric reference. Because the bar would not be moved, it was safe to assume that the origin would stay fixed and the robot would always know how to return to it, as well as reference all other locations from the bar's location. To prevent the accumulation of drift from odometry inaccuracies, the turtlebot would reset the home base coordinate whenever it returned to the bar (homing in on the AR Tag). Whenever the turtlebot took an order, it would also log the odometric coordinate of the customer so that returning to the customer from the bar would not be a random walk through the crowd.

Because all of the coordinates were relative to a single origin, there was no absolute grid placed over the active area. Thus, whenever the robot wanted to return to a known stored location (such as returning to the bar after taking sufficient orders, or returning to a customer after picking up their order from the bar) the turtlebot would first check its angle and compare it to a calculated angle between its current coordinate and the goal coordinate. It would then use this data in order to rotate so that it was facing in the direction of the goal location. Then it would proceed to move in that direction (while prioritizing obstacle avoidance via its IR proximity camera) until its current odometric coordinate was within 0.2 meters of the goal coordinate. It would then switch states to the "homing on AR tag" because it was now located at a close enough distance to rely on AR tag navigation.

Homing on an AR Tag:

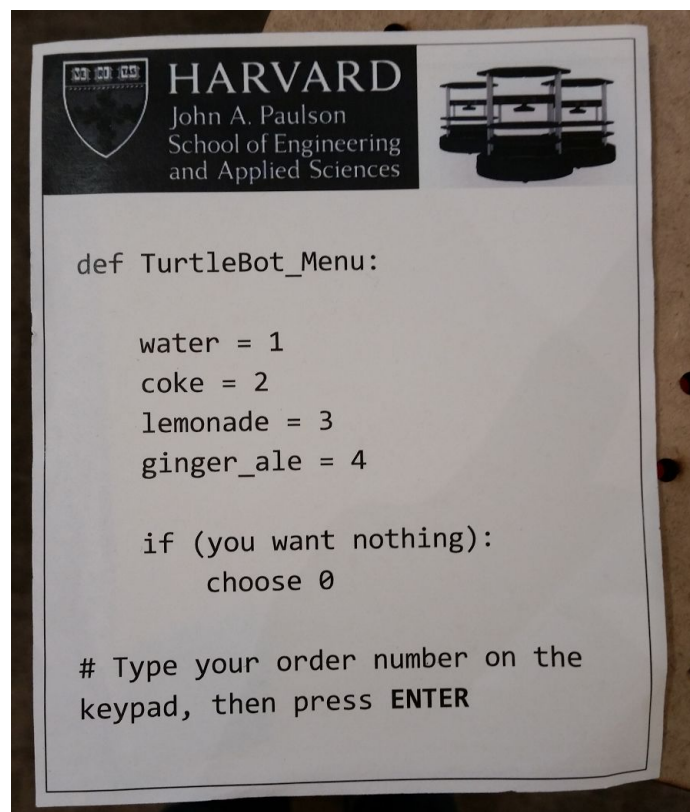
Within distances of about 1 meter, the turtlebot would be able to accurately navigate to an AR tag and face it directly. The general process was to retrieve the x and z coordinates of an identified tag, then try to move the bot so that the x coordinate is within a center threshold of the image frame, all the while moving towards the tag until the z coordinate (distance from the bot) is within a certain z coordinate threshold. If more than one tag was spotted, the bot would prioritize the first tag. It would always avoid approaching the Bar's AR tag (hard-coded as id = 0)

unless it was specifically searching for it because it was returning to the bar to request drinks. It would also avoid tags if the tag were already associated with an outstanding order.

As stated before, the homing state would only ever follow a wander state, or a navigate-to-coordinate state, because AR recognition at large distances was poor.

Taking Orders:

When the turtlebot had approached within a certain threshold distance of a tag while it was taking orders, it would stop and ask the guest to place an order. The robot would direct the guest to input an order on the keypad and press enter. If the guest took too long, it would prompt him/her to hurry up, before eventually just leaving. The guest also had the option to input 0 if they did not want to place an order. Any invalid keypad syntax would give the guest an option to try again once before abandoning the order. When the guest ordered, the turtlebot would store the AR Tag ID, the item ordered, and the location of the order in order to find the guest again. It would then verify that it would return soon with the item, and request that the guest stay in the same location.



The menu that was displayed on the turtlebot.

When the turtlebot returned to the bar, it would parse the orders that had been placed into natural language (e.g. "I need two cokes, one ginger ale, one water) and request that the bartender supply its tray. The bartender would then use the keypad to confirm the order had been loaded, and the turtlebot will set off again.

Delivering Orders:

The turtlebot will deliver orders in the order in which they were received. It will use the stored location to approach the area in which the order was taken (see returning to a known location). When it sees the correct AR tag for the order it will home in on it (see homing on an AR Tag). When it falls within a threshold distance of the guest, it will prompt them to take the item they had ordered and ask for a keypad confirmation to ensure that he/she has taken it. It will then proceed to deliver the next order, or go back to taking orders if all orders have been fulfilled.

Results and Success

Our robot was fairly successful in completing its task in an idealized environment with people who knew how to use it effectively. The turtlebot could effectively take multiple orders, return to the bar, pick up the correct orders, and manage to locate the people who placed them provided they remained in the same general area. At the SEAS design fair, however, there were several obstacles to proper performance. These included the guests' tendency to wave their AR tags in front of the robot's camera, which made it much more difficult to sense than if the tag were held perfectly steady at a distance. The speaker was also very difficult to hear amongst the noise, so operation of the waiter required one of the designers to explain to the guest.

The small and crowded setting also made our design seem unnecessarily complicated. In the context of the robot party at the SEAS fair, a robot that walked around with a fully loaded tray of drinks for guests to take probably would have been more useful. With guests constantly coming and going, few people had time to stand and wait for their drink to be delivered. However, we did not believe that this offered a significant enough design challenge. We believe that the concept of our waiter robot could be extrapolated to serve drinks and even meals at a restaurant or a more formal, static gathering. The setting we envision would be a restaurant or a ballroom with an AR tag above each table, where people tend to remain seated until they are served food.