Marco Polo

World Renowned Intrepid
Adventuring Robot

ECE 578 Fall 2019

Josiah Sweeney, Robert Holt, Christopher Neighbor

Overview/Goals/Requirements

- Marco Polo can localize and map simultaneously.
- Marco Polo can move autonomously.
- Marco Polo can recognize a red cube.
- Marco Polo can detect and display relative distance to recorded objects.
- Marco Polo can respond via speech synthesis.
- Marco Polo can transcribe words spoken to him.
- Marco Polo can abandon exploration and move to the found object.
- Marco Polo can pick up the found object.

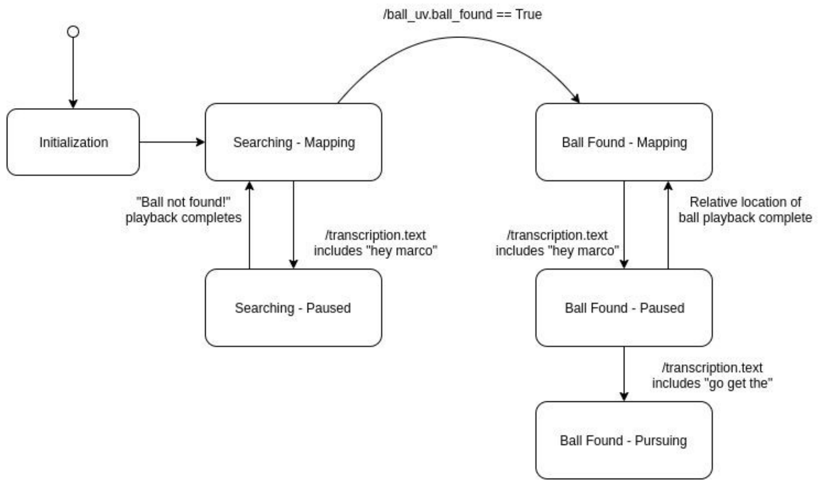
ROS Project

Marco Polo was split up into several nodes:

- “Marco Overlord” state machine
- RTAB-Map SLAM
- Frontier Exploration
- Object Recognition
- Object Depth Finding
- Speech Synthesis
- Speech Transcription
- Movement Manager
- Arm Control

Marco Polo State Machine

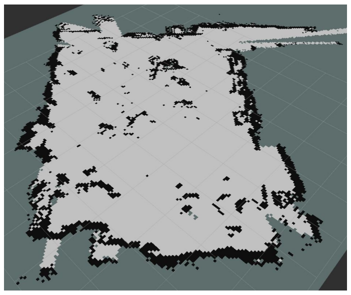
- Assumes that cube doesn't change locations.
- Relies on user addressing Marco before asking about the location of the ball


/ball_uv.ball_found == True
Initialization
Searching - Mapping
Ball Found - Mapping
"Ball not found!" playback completes
/transcription.text includes "hey marco"
/transcription.text includes "hey marco"
Relative location of ball playback complete
Searching - Paused
Ball Found - Paused
/transcription.text includes "go get the"
Ball Found - Pursuing

Mapping/Localization

In Marco Polo, we utilized RTAB-Map

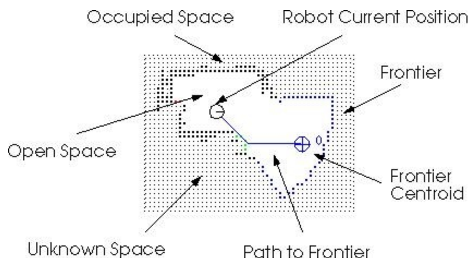
- RTAB-Map is a specific GraphSLAM approach to SLAM that is based on an incremental appearance-based loop closure detector.
- The loop closure utilizes a “bag-of-words” approach to determine how likely a new image comes from a previous location or a new location.
- When a loop closure hypothesis is accepted, a new constraint is added to the map’s graph, then a graph optimizer minimizes the error in the map.



Autonomous Exploration

In this project, a frontier based autonomous exploration approach was integrated into the robot.

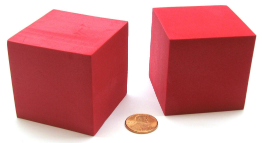
- Frontier based autonomous exploration means that the robot is given a “frontier”, or unexplored area to explore and map out.
- This is achieved with Marco Polo by “drawing” a polygon in RVIZ on top of the area that Marco polo is in.
- The polygon is created by clicking on the screen creating a series of connected points until the shape is closed.

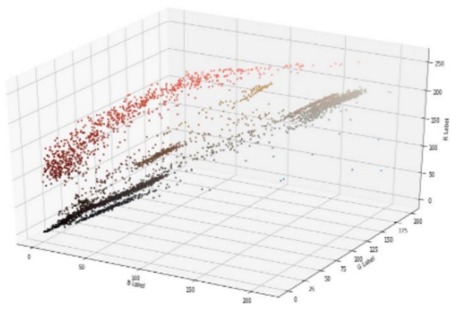


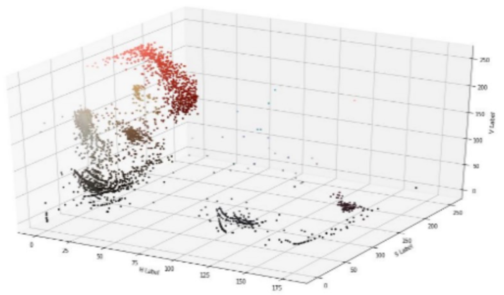
Objection Recognition

Steps:

- Capture video frame from ROS topic node
- Convert to OpenCV image using cv_bridge. Frame is now 3D numpy array in BGR
- Convert image to HSV values for improved differentiation and clustering using cuboids



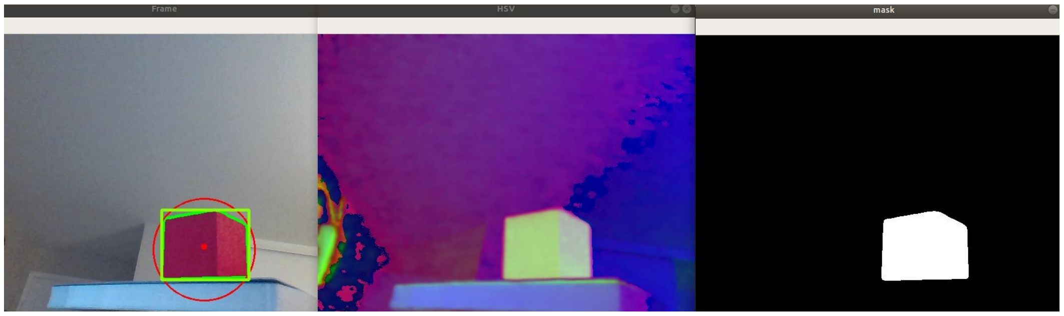




Objection Recognition

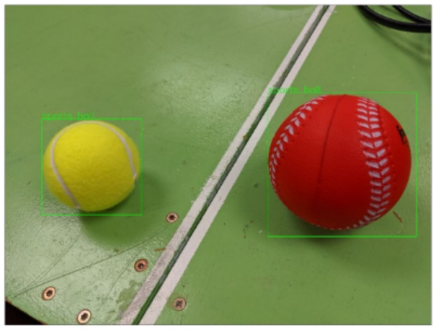
Next Steps:

- Filter for red hues which are not extremely light or dark.
- Check object to see if it is big enough and squarish in shape
- If red circular object detected, publish the coordinates of the center of the cube, ex. [1, 320, 240]

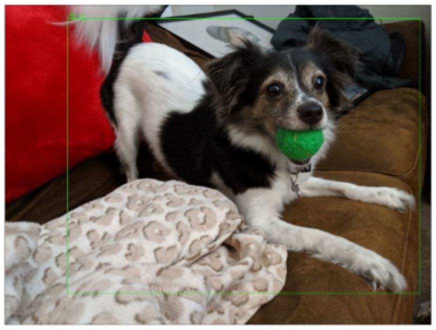


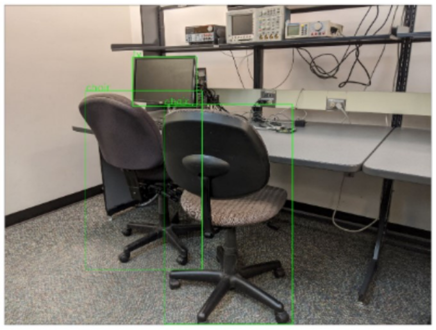
Objection Recognition: Deep Learning

Future work could involve using a PyTorch model which has been trained on the COCO object segmentation dataset for localization and detection of objects. This makes the model more robust than merely detecting objects by their color and shape.
These labels and location could be sent to the robot and be incorporated into the speech and mapping. Issues with speed, memory, and confidence values will need to be resolved.



STAY!

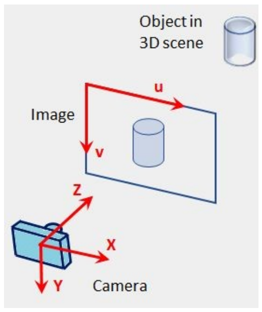




Object Depth Finding

Marco Polo looks for the cube and attempts to add the ball to the world frame.

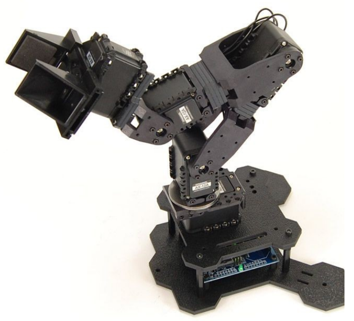
- This is achieved by receiving a center-point pixel value from the object recognition node, using that pixel value U,V to map it to a XYZ value from a pointcloud generated by the RGB-D camera on the robot.
- The XYZ values are transposed from the frame of the robot's world pose at the time of the ball object recognition.
- This is done utilizing the transform library in ROS, which stores and calculates the transforms of different robot objects, stationary and mobile, storing their relative translation and quaternion rotation transforms.



Movement Manager

- The second half of the project includes the creation of the “movement manager” ROS node which brought autonomy to the mapping process, and more importantly, handles the transition to a pursuit mode of the cube.
- The movement manager has an interface for receiving a “pursuit request” from the Marco Overlord. When this request comes in, the current Frontier Exploration task is aborted via the Frontier Exploration SimpleActionClient.
- Once the position determined above has been reached (i.e. the move_base action server indicates that the goal is complete) , the arm actuation takes place and the pickup attempt occurs.

Robot Arm Movement



- In the second half of the project, we added in the capability of the robot to grab the sensed object in front of it, based on the dead reckoning of the robot's movement toward the sensed object.
- This was achieved by adding in functionality from the phantomx arm library and programming a node in ROS to move each joint when prompted when the robot reached its goal.
- If we had more time, improvements could have been done on the robot to increase the reliability of the arm. We could have incorporated the second camera to actively adjust the servos on the arm to grab to cube, allowing it to adjust to the innactuarace and drift of the robot after it approaches the cube

Speech Synthesis

- Utilized Amazon Polly API due to scalability, ease of use, and fast synthesis
- Synthesis performed for location of the ball when Marco is addressed verbally
- Synthesis based on two options:
  - Standard TTS - Phoneme concatenation
  - Amazon "Neural TTS" - Sequence of spectrograms created via neural network which uses sequence of phonemes as input.
    - Adds context awareness from trained nuance of speech

Speech Transcription

- Utilized Google Cloud Speech-to-Text due to scalability, live streaming capability, and confidence/stability tradeoff metrics
- Examples
  - **results** { alternatives { transcript: "tube" } stability: 0.01 }
  - **results** { alternatives { transcript: "to be a" } stability: 0.01 }
  - **results** { alternatives { transcript: "to be" } stability: 0.9 } results { alternatives { transcript: " or not to be" } stability: 0.01 }
  - **results** { alternatives { transcript: "to be or not to be" confidence: 0.92 } alternatives { transcript: "to bee or not to bee" } is_final: true }
  - **results** { alternatives { transcript: " that's" } stability: 0.01 }
  - **results** { alternatives { transcript: " that is" } stability: 0.9 } results { alternatives { transcript: " the question" } stability: 0.01 }
  - **results** { alternatives { transcript: " that is the question" confidence: 0.98 } alternatives { transcript: " that was the question" } is_final: true }

Challenges

- Gathering the required parts took days, and some parts, such as a USB Hub and wiring to the Kobuki base were missing, along with several necessary power adapters.
- Linux kernel on the Joule was corrupted and necessitated a full build-up of the robot from the ground up.
- Lack of a combination of internet connection and ability to ssh into Marco Polo.
- Different flavors of ROS and Ubuntu and joining code.
- State of the Robot Arm (incorrect baud rate and false hardware limits)
- More missing parts for project 2 including FTDI cable
- Size of the object in project 2 made finding the cube VERY challenging.

Lessons Learned

- The ROS framework offered flexibility in interfaces which enabled modular development
- Reliance on PSU WiFi infrastructure is challenging, PSU CAT is unresponsive or unwilling to help with challenges
- Late integration caused inevitable unforseen misunderstandings
- The robot cannot reliably enough dead reckon for the robot arm.
- More filters on the object recognition will allow for a more robust finding of the smaller cube.