School of Engineering | UNIVERSITY OF St.Thomas

Navigation and Manipulation of a Camera Integrated ROS-Based Robot

Cheol-Hong Min
Center for Intelligent Robotics and Autonomous Systems (CIRAS)
Electrical and Computer Engineering
University of St. Thomas, St. Paul, MN

Agenda

- Overview and Objectives
- Subsystem Components
  - Mobile Robot Platform
  - Actuator Arm
  - Depth Camera
- Integration
- System Components
- Demonstration Videos

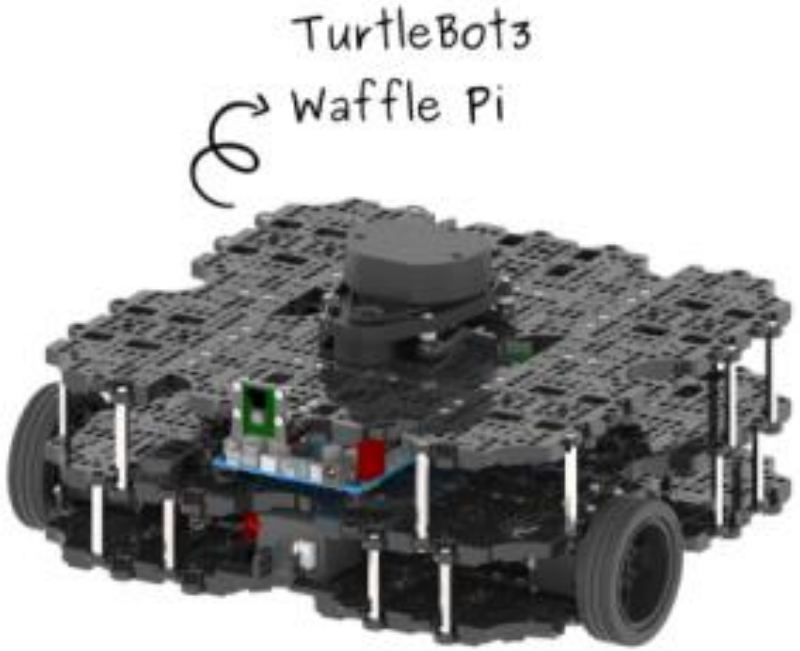
Overview

- Objective 1
  - Develop an autonomous mobile robot using machine vision to provide autonomous navigation, object recognition, and object interaction
  - This project would consist of a ROS-based mobile robot, a robotic arm mounted to the robot, an integrated depth camera, and a Raspberry Pi processor
- Objective 2
  - Successful development of machine vision algorithms coupled with Lidar for navigation and object recognition for pick and place service robot

Turtlebot3 Mobile Robot

- Turtlebot3 Used as Mobile Robot

- Turtlebot3 Features
  - 11” x 12” x 5.5”
  - Differential Drive
    - Dynamixel Motors
  - LiDAR
    - LDS-01
  - Raspberry Pi 3
  - OpenCR Board
  - Raspberry Pi Camera

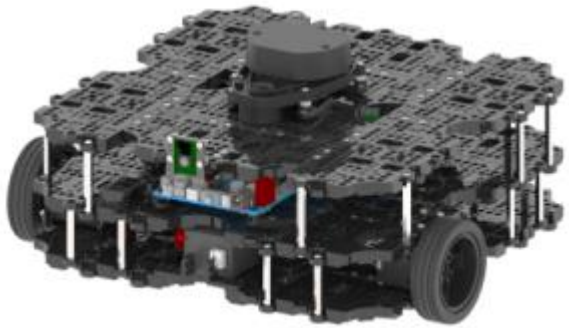


OpenMANIPULATOR-X Arm

- OpenMANIPULATOR-X Arm Used as Actuator

- OpenMANIPULATOR-X Features
  - Dynamixel Motors
  - 15” reach
  - 46 RPM joint speed
  - 1.54 lbs
  - Compatible with ROS
  - Controlled via OpenCR

Mobile Robot Platform



Turtlebot3

+

OpenMANIPULATOR-X Arm

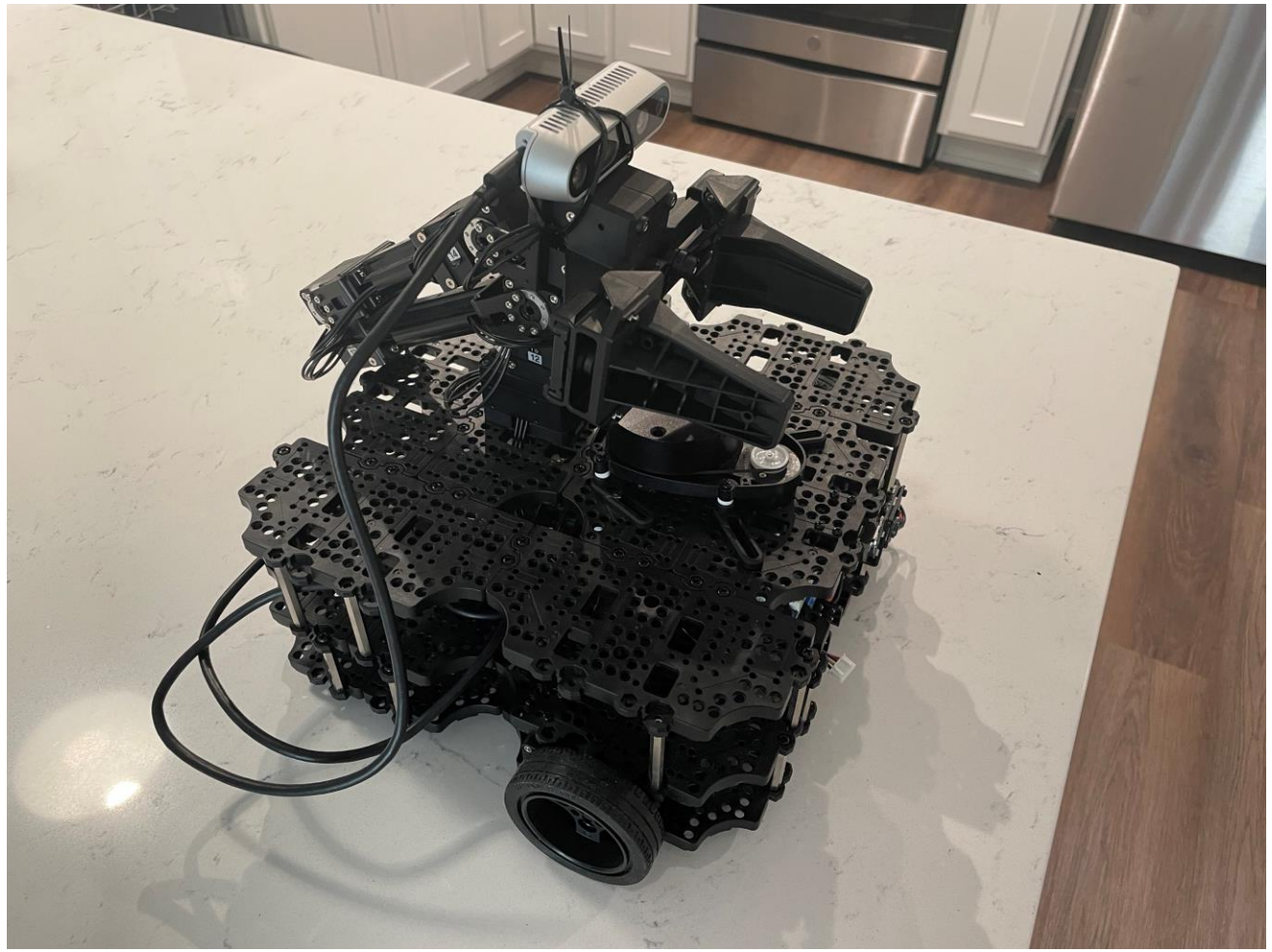
→

Camera

- Intel RealSense Depth Camera D435i

- Features
  - Color Camera
  - Depth Camera
  - USB Interface

Robotic Platform with Camera



Turtlebot3 and Camera Integration

OS Integration

- It is recommended to install Ubuntu onto Turtlebot3 Raspberry Pi
- Camera OS compatibility
  - Ubuntu 22.04
  - Raspbian
- Integrating with Ubuntu 22.04 on Raspberry Pi caused difficulty
  - Required “combining” installation instructions of Ubuntu and Raspbian
    - Ex. Swap file size changes
- Low Memory Issue (upgrade to 64GB SD card or higher: 128 rec)

Operating System Configurations

- Remote Host
  - Linux Machine
  - Ubuntu 22.04
  - ROS2
  - ROS2 Humble

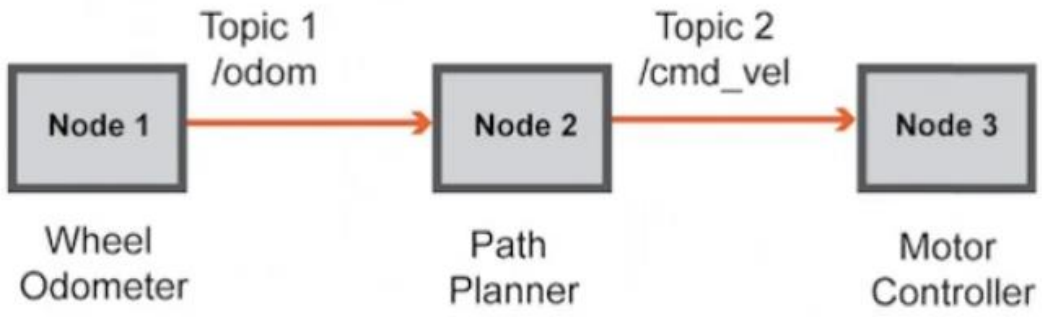
- Turtlebot3
  - Raspberry Pi
  - Ubuntu 22.04
  - ROS2
  - ROS2 Humble

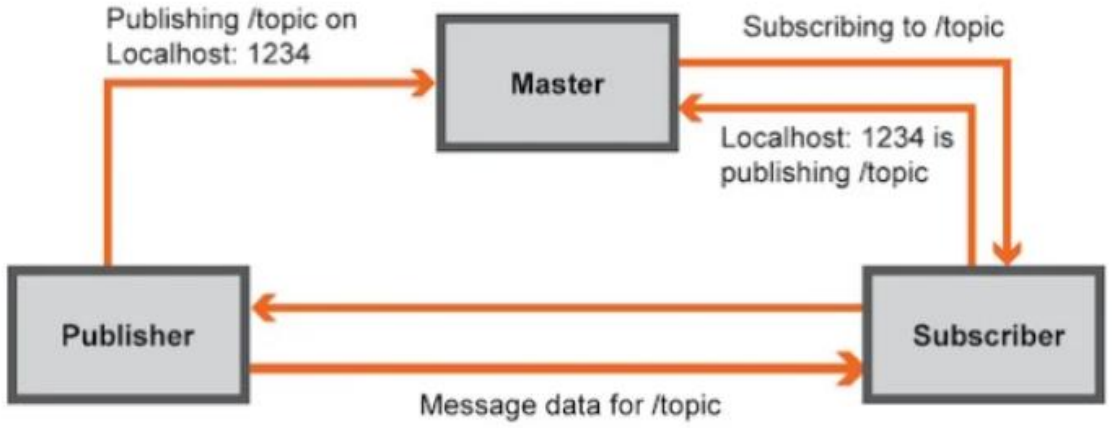
Robotic Operating System (ROS) Overview

- Operating System framework providing
  - Hardware abstraction
  - Device control
  - Communication
  - Package management
- ROS
  - on Linux
  - on Virtual Machine

Robotic Operating System (ROS) Overview

- Nodes
  - Use Topics for Inter-communication using Publisher/Subscriber model





Software Integration

- Depth camera and object recognition packages installed onto Turtlebot3 Raspberry Pi 4
- Depth camera software integrated into the launch file
- ROS publisher node created to communicate depth camera information to the Remote PC

Turtlebot3 Setup

Remote Host

ROS Communication Link

Turtlebot3

Launching Mobile Robot

- Running ‘roscore’ on Remote PC:
  - ‘roscore’ is a collection of nodes and programs that are prerequisites of a ROS-based system

Remote Host

- ‘roscore’ provides:
  - ROS Master
  - ROS Parameter Service

What is a Launch File?

- File executed in ROS to spawn nodes

autonomous_explorer.launch
~/catkin_ws/src/ros_autonomous_slam-master/launch

```
<launch>

  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="move_forward_only" default="False"/>
  <arg name="explorer" default="explore-lite"/>
  <arg name="map_file" default="$(env HOME)/catkin_ws/src/ros_autonomous_slam-master/maps/my_map.yaml"/>

  <!-- Turtlebot3 Manipulation Bringup -->
  <node pkg="turtlebot3_manipulation_bringup" type="turtlebot3_manipulation_bringup" name="turtlebot3_manipulation_bringup" output="screen"/-
>

  <!-- Custom Robot Navigation Python Script -->
  <node pkg="warmup_project" name="custom_turtlebot_nav_stack" type="custom_turtlebot_nav_stack.py" output="screen"/>

  <!-- SLAM Node -->
  <include file="$(find ros_autonomous_slam)/launch/turtlebot3_slam.launch">
  </include>

  <!-- move_base -->
  <include file="$(find turtlebot3_navigation)/launch/move_base.launch">
    <arg name="model" value="$(arg model)" />
    <arg name="move_forward_only" value="$(arg move_forward_only)"/>
  </include>

  <!-- move_group -->
  <include file="$(find turtlebot3_manipulation_moveit_config)/launch/move_group.launch">
  </include>

  <!-- Autonomous Explorer Node -->
  <include file="$(find ros_autonomous_slam)/launch/$(arg explorer).launch">
  </include>


/launch
```

Launching Mobile Robot

- Launch file on Turtlebot3 (ssh remote access via remote host)
  - turtlebot3_core
  - turtlebot3_lidar
  - turtlebot3_diagnostics
  - Depth camera master_file.py

Turtlebot3

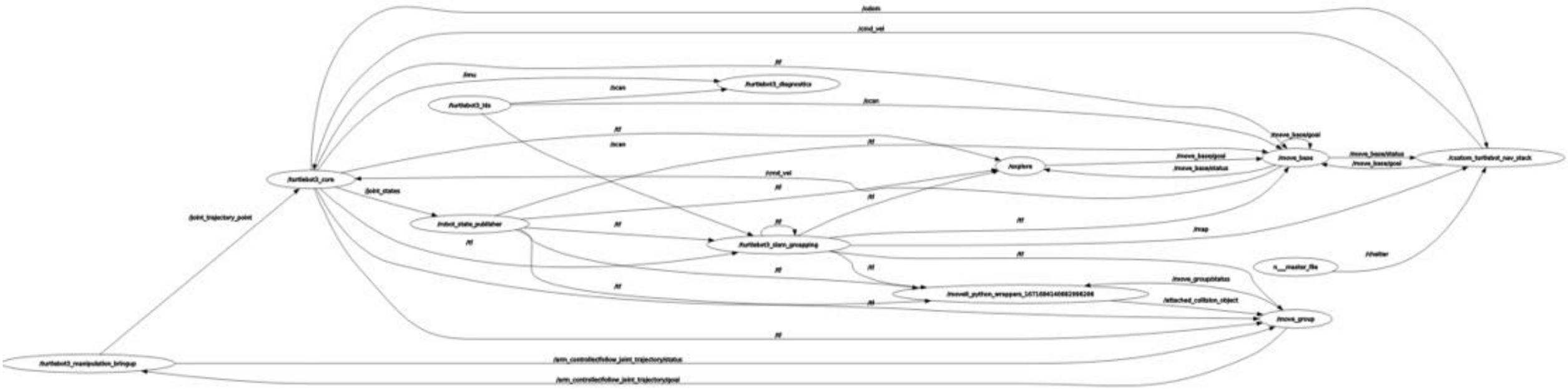
Launching Mobile Robot

School of Engineering | UNIVERSITY OF St.Thomas

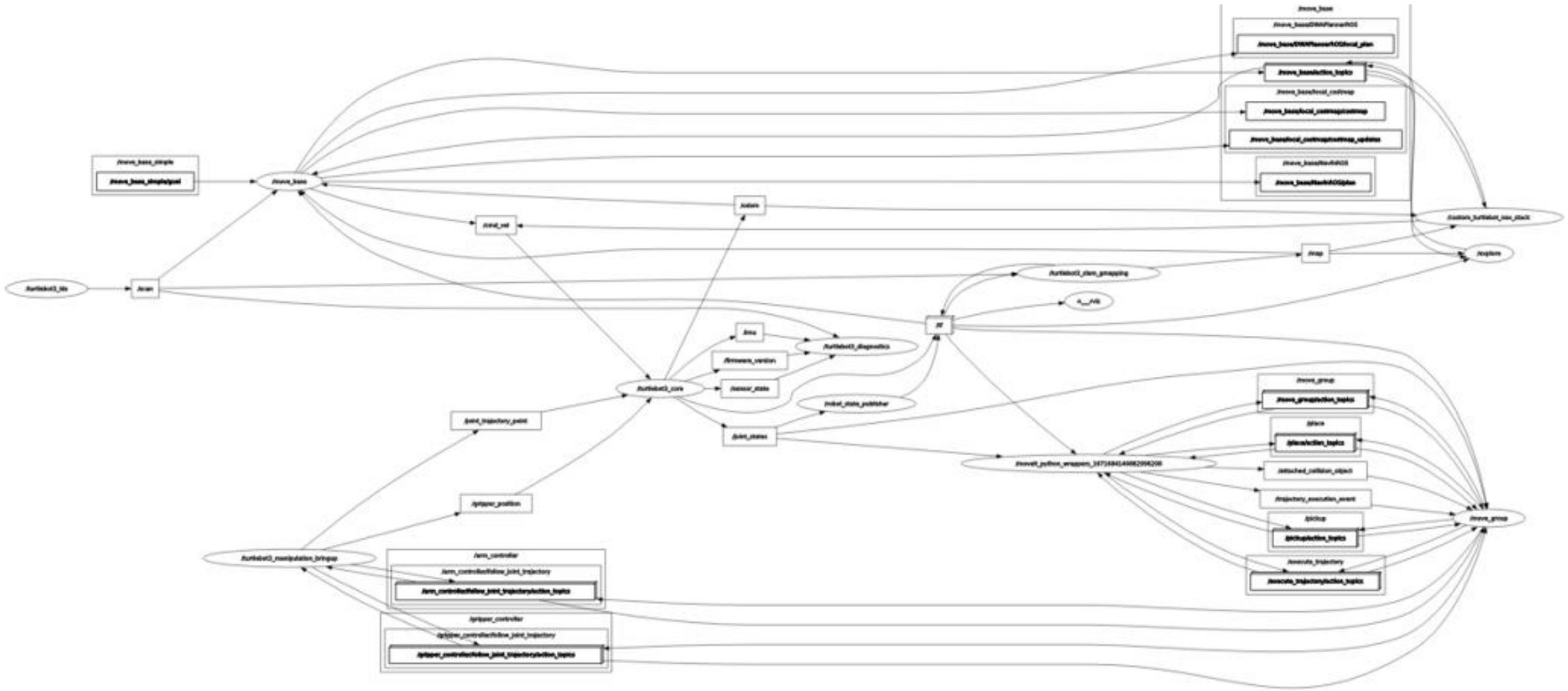
- Launch file on remote PC
  - turtlebot3_manipulation_bringup
  - turtlebot3_slam
  - move_base
  - move_group
  - autonomous_explorer
  - Custom navigation curstom_turtlebot_nav_stack.py

Remote Host

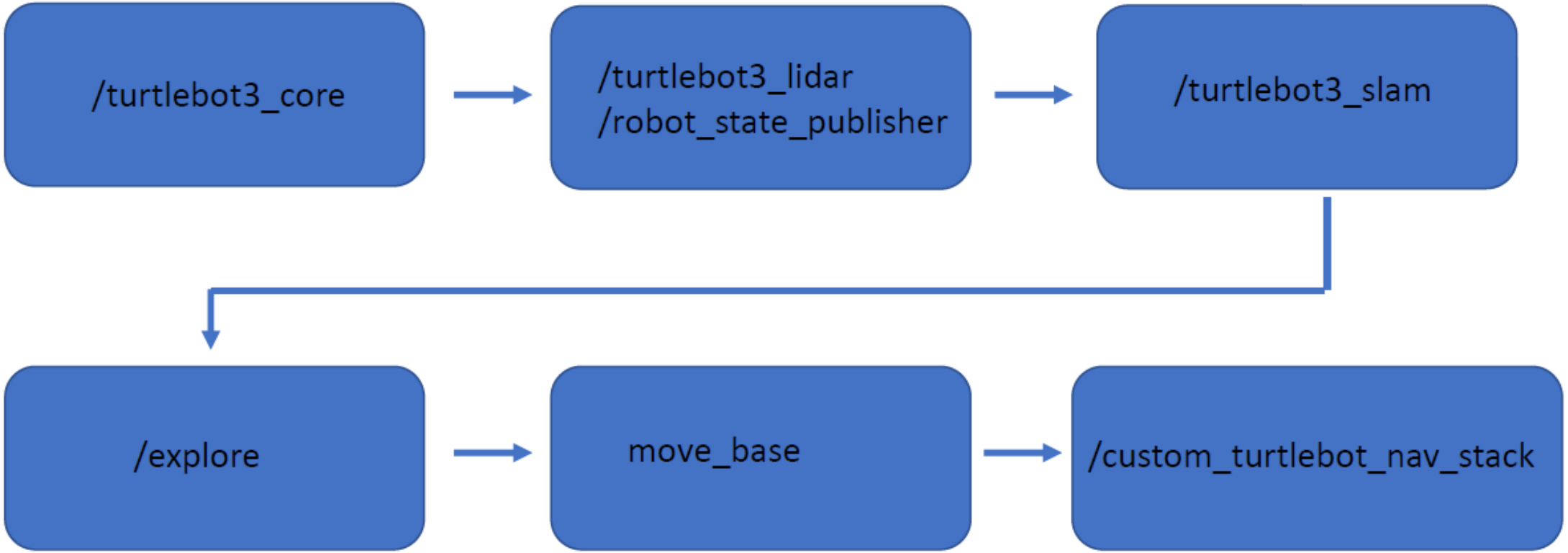
System Nodes



System Nodes and Topics

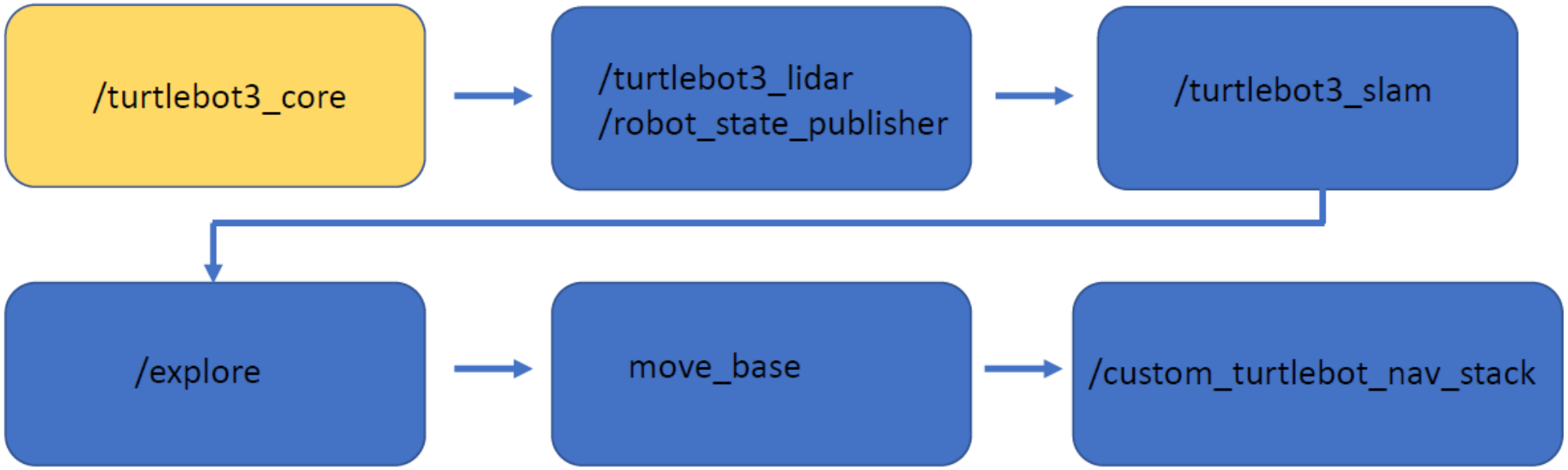


Navigation - Nodes and Topics



/turtlebot3_core

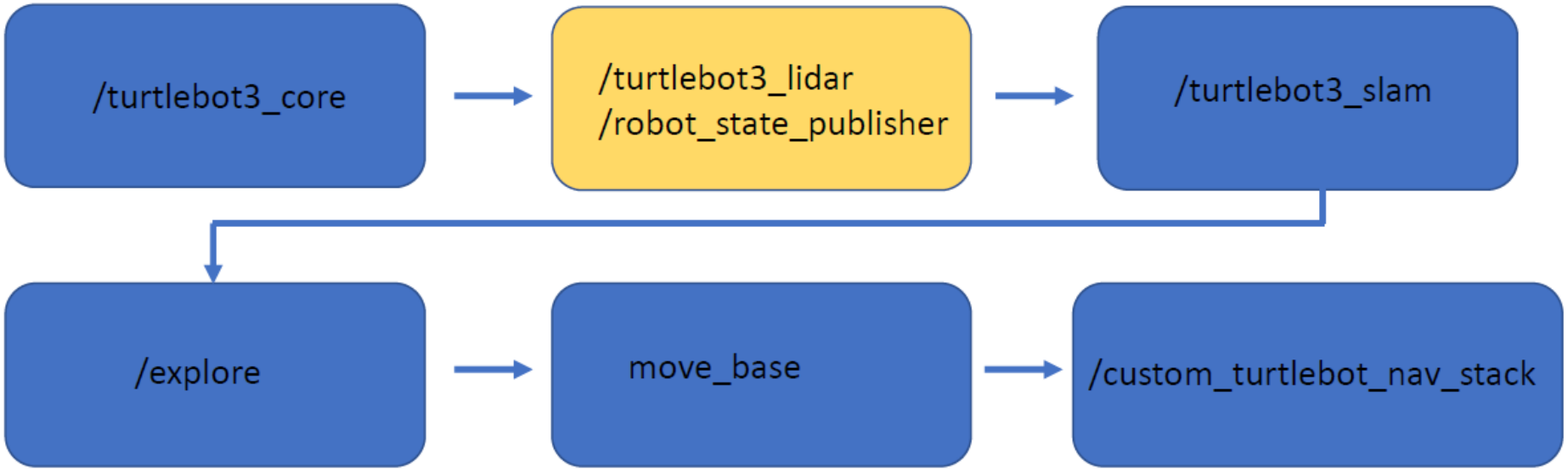
- Subscribes to /cmd_vel
  - Control the translational and rotational speed of the robot
- Subscribes to /motor_power
  - Torque control
- Subscribes to /rest
  - Reset odometry and IMU data
- Subscribes to /sound
  - Output sound



/turtlebot3_lidar & /robot_state_publisher

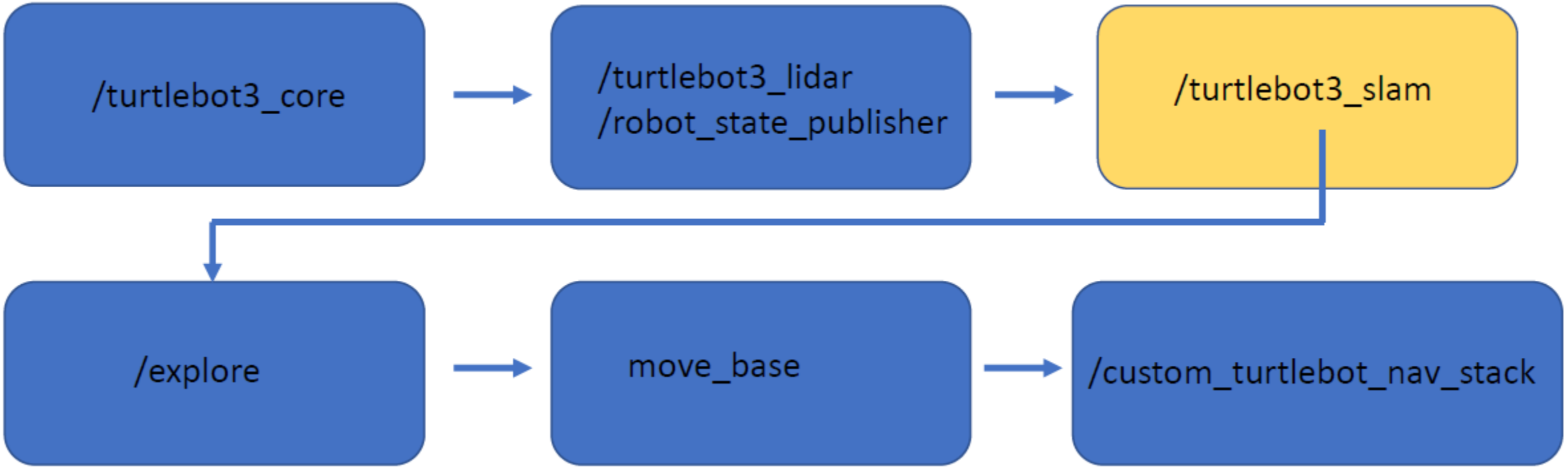
School of Engineering | UNIVERSITY OF St.Thomas

- Publishes /scan
  - LIDAR scanner data
    - Angles
    - Time increment
    - Range
    - intensity
- Publishes /joint_states
  - Also used with an actuator
  - Translation between robot positioning and LIDAR view



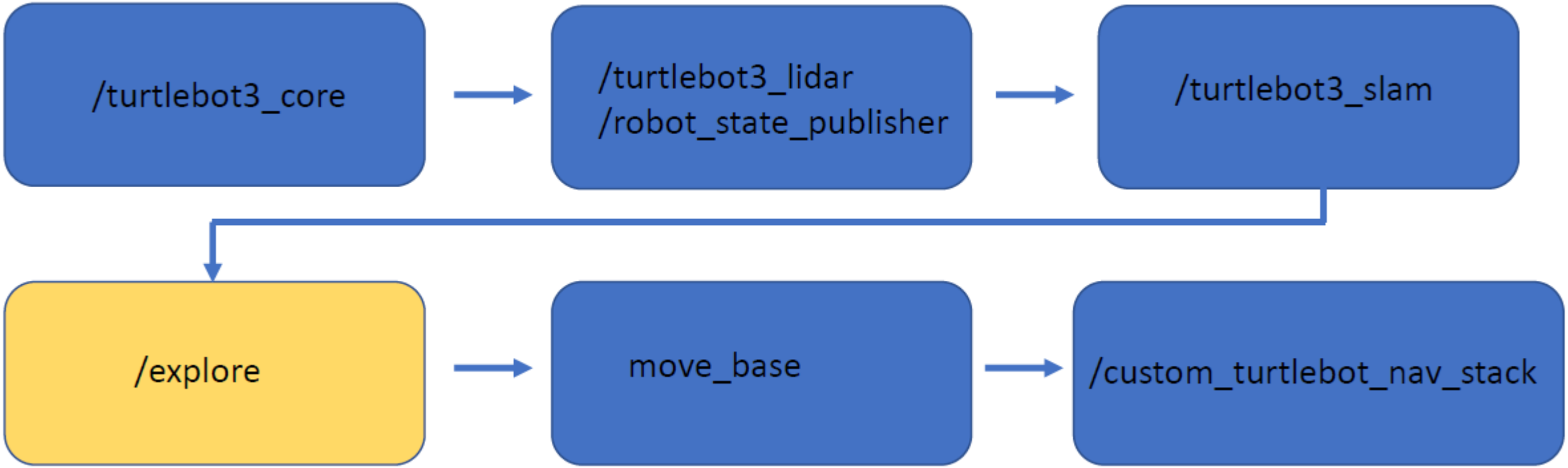
/turtlebot3_slam

- Multiple topics related to base_frame, odom_frame and mapping
- Combines sensor data (odometry, IMU, LIDAR, encoders) to produce a map of the environment



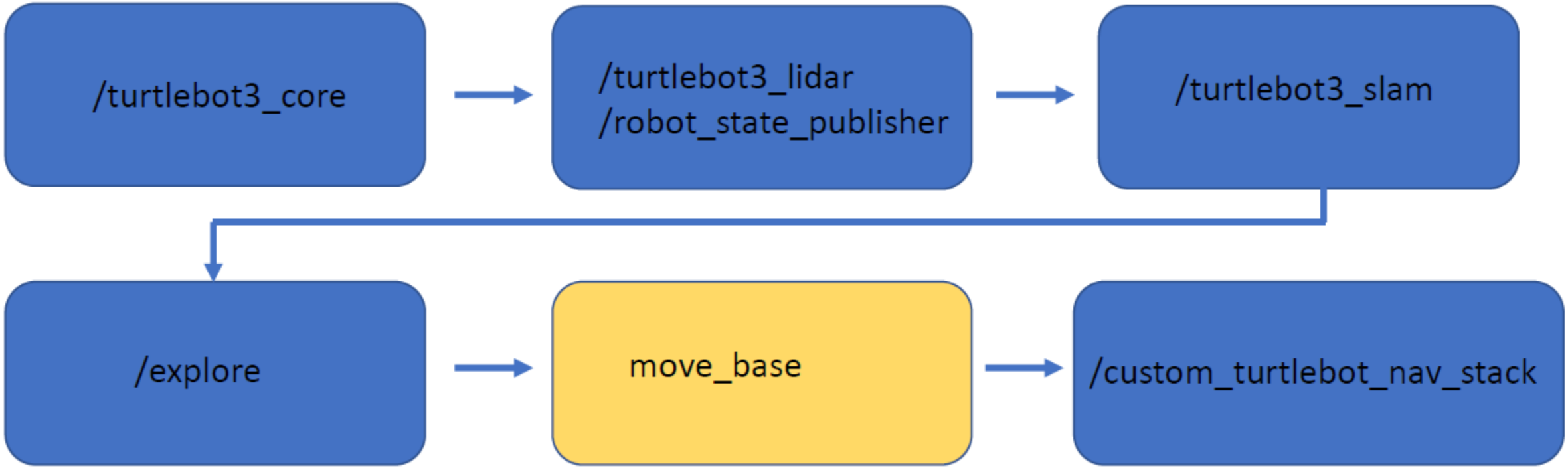
/explore

- Frontier-base exploration of an unknown map
- Utilizes map produced by turtlebot3_slam to provide robot commands through move_base node
- Provide for autonomous exploration



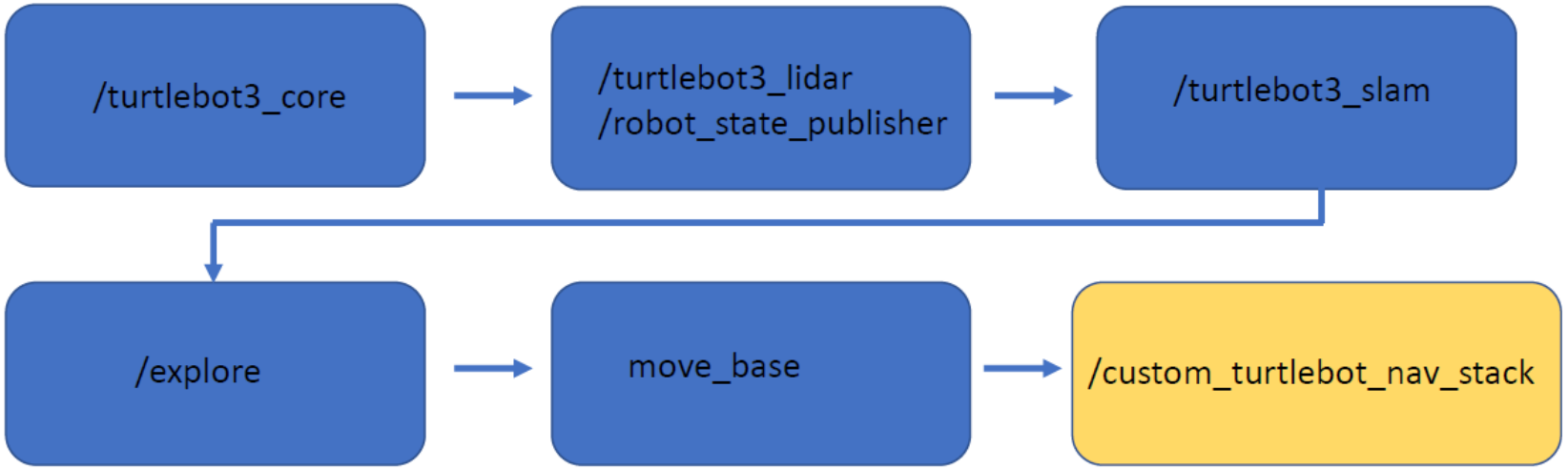
/move_base

- Subscribes to /move_base_simple/goal
  - This goal is output by /explore node
  - Provides a target position for the robot to achieve
- Publishes to /cmd_vel
  - Stream of velocity command to be executed by mobile robot



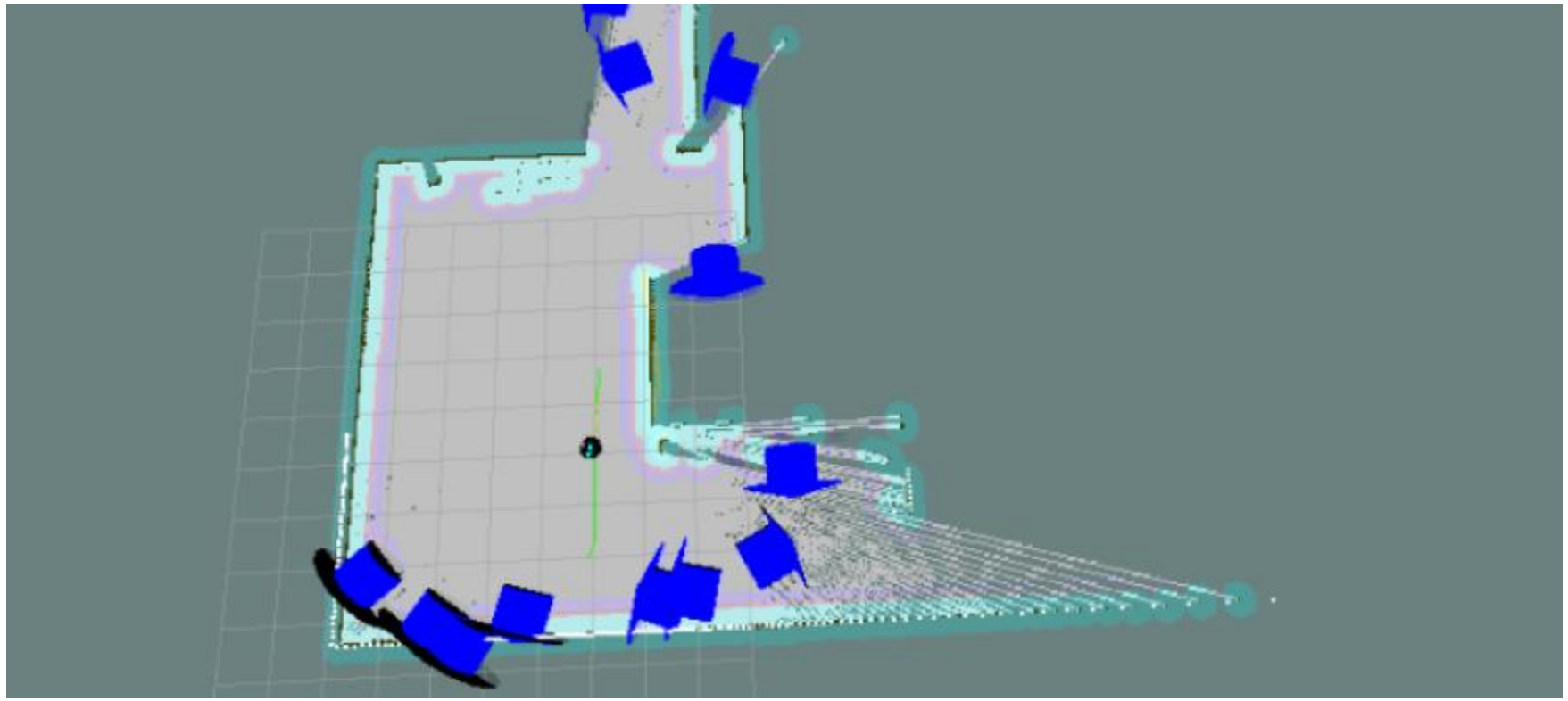
/custom_turtlebot_nav_stack

- Subscribes to /explore
  - Receives progress and status of autonomous exploration
- Subscribes to /map
  - /map is the output of SLAM
  - Map data necessary to instruct robot to navigate to position on map
- Subscribes to /odom
  - Odometry data necessary to translate relative robot positioning to global frame
- Provides simple action client to /move_base
  - Allows for instructions to be sent to /move_base to move robot

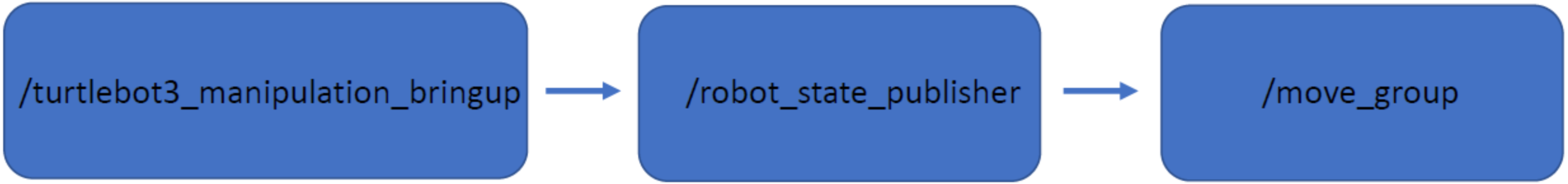


Navigation Stack

- Ties in most navigation nodes for a simpler command interface
- Takes in information from odometry, LIDAR, goal positions, and costmaps to output safe velocity commands within the SLAM

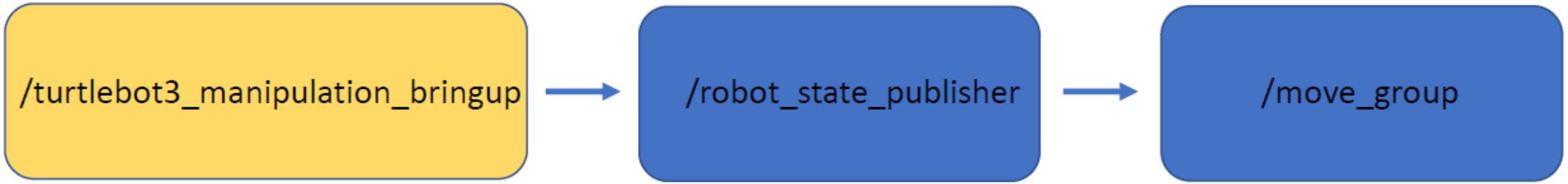


Actuator - Nodes and Topics



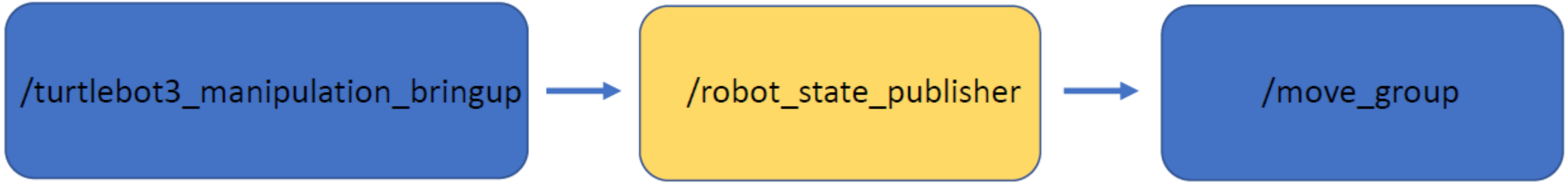
/turtlebot3_manipulation_bringup

- Parameters and settings for OpenMANIPULATOR-X Arm to interact with Turtlebot3
  - Number of joints
  - Joint lengths
  - Joint rotation limits
  - Gripper width



/robot_state_publisher

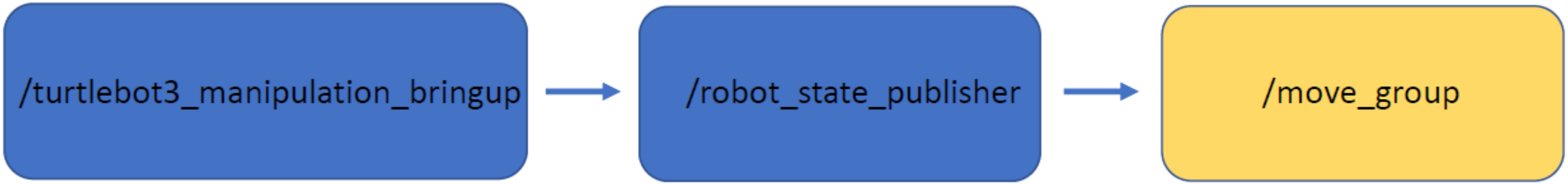
- Publishes /joint_states
  - Getting current joint values to translate to position
  - Setting current joint values
  - Planning arm paths
  - Actuating gripper



/move_group

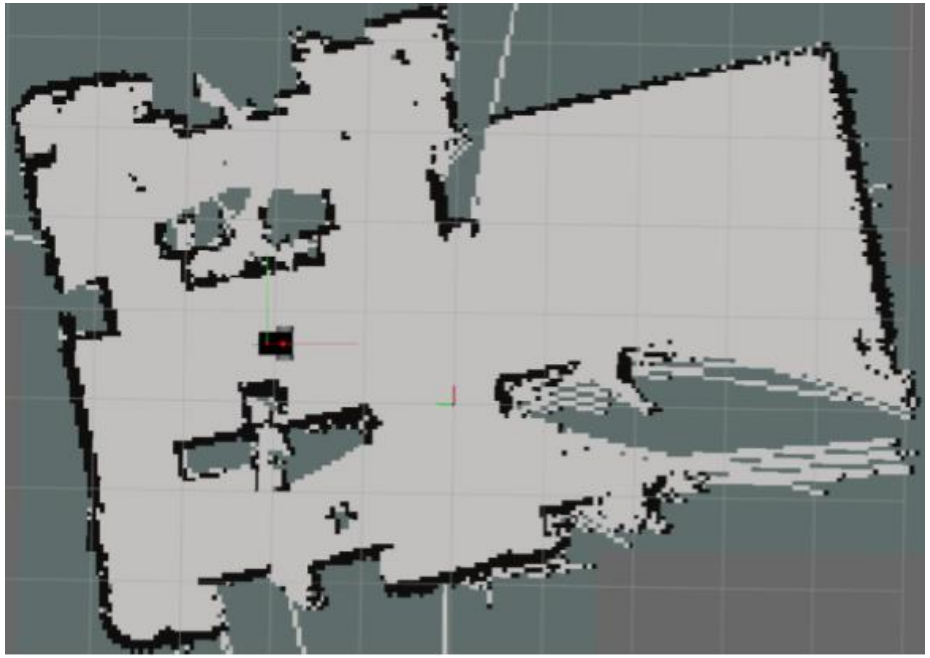
- Inputs
  - Joint state information
  - Transform information
  - Controller interface
  - Motion plan

- Using inputs, /move_group performs arm and gripper actuation with collision avoidance



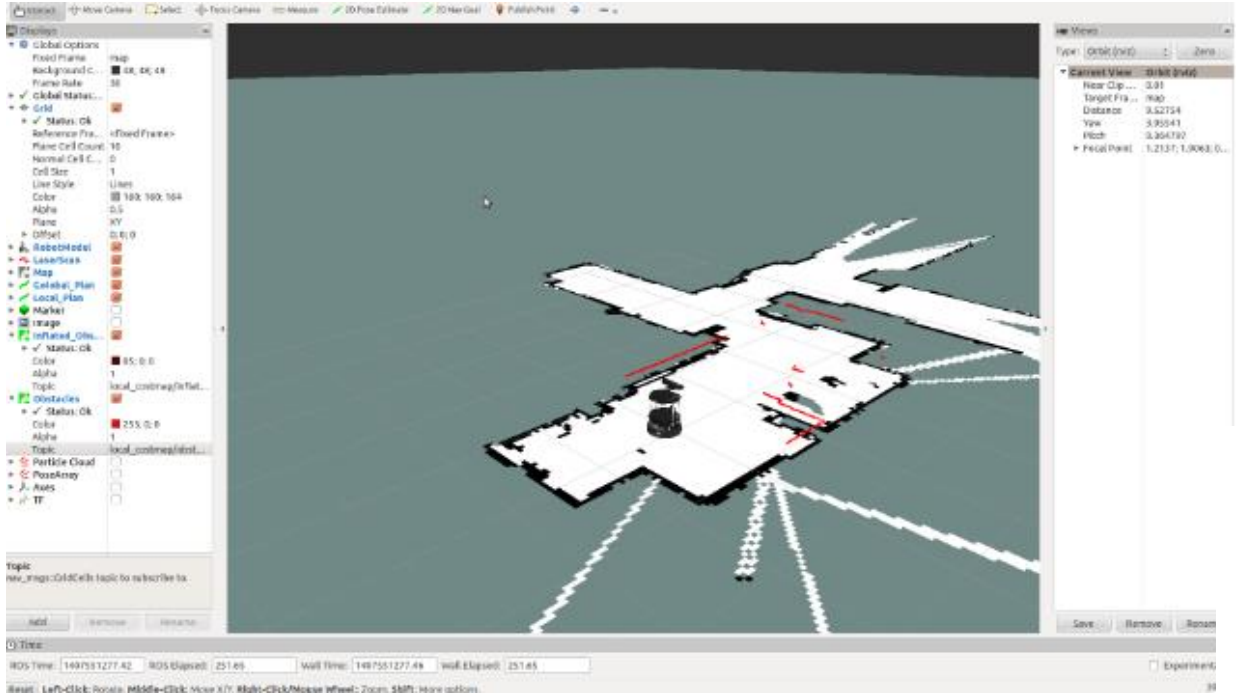
Localization and mapping

- SLAM
  - Simulates localization and mapping
  - Uses sensors to gather environment information
    - LIDAR and odometry
  - Costmaps
    - “cost” of traversing different areas
    - Global: long-term plans
    - Local: short-term plans (obstacle avoidance)
  - SLAM constantly publishes the map
  - Map continuously updates while SLAM is active
  - Network integrity and speed is important



Robot Visualization

- RViz
  - Robot Visualization
- Graphical Interface tool showing what robot “sees” and its environment



Autonomous Exploration

- Utilizes /explore navigation node to explore unknown map frontiers
- Does not create its own map but rather subscribes to map generated by SLAM and finds unexplored frontiers to navigate to
- This algorithm eliminates the need for initial robot localization
  - Robot can be placed at a random position in an environment and map out the entire area with no user intervention
    - Able to do so by creating a global map based off robot's initial relative position
    - Other algorithms require user intervention to

Robot State Machine - custom_turtlebot_nav_stack

School of Engineering | UNIVERSITY OF St.Thomas

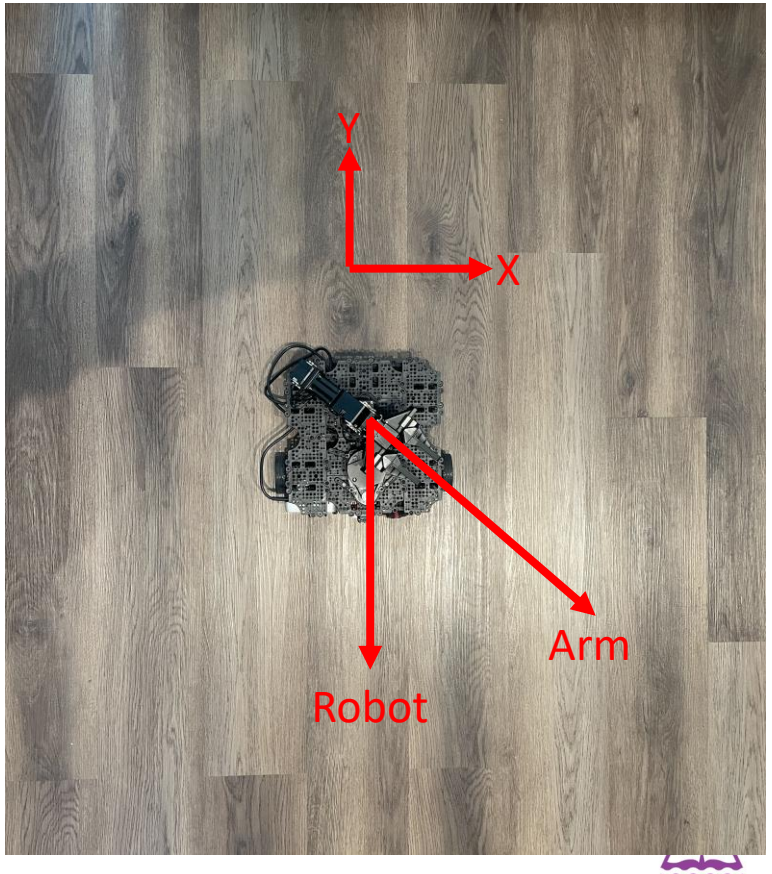
- Initialize arm position
- Perform autonomous exploration
- Upon autonomous exploration completion, search for the target object class and width
- Determine the global position of the target object
- Drive to object
- When the robot reaches 0.15 meters away from the object, stop and re-read the object position
- Determine the global position of the target object
- Extend the arm to the object
- Actuate gripper
- Repeat

Robot Drive

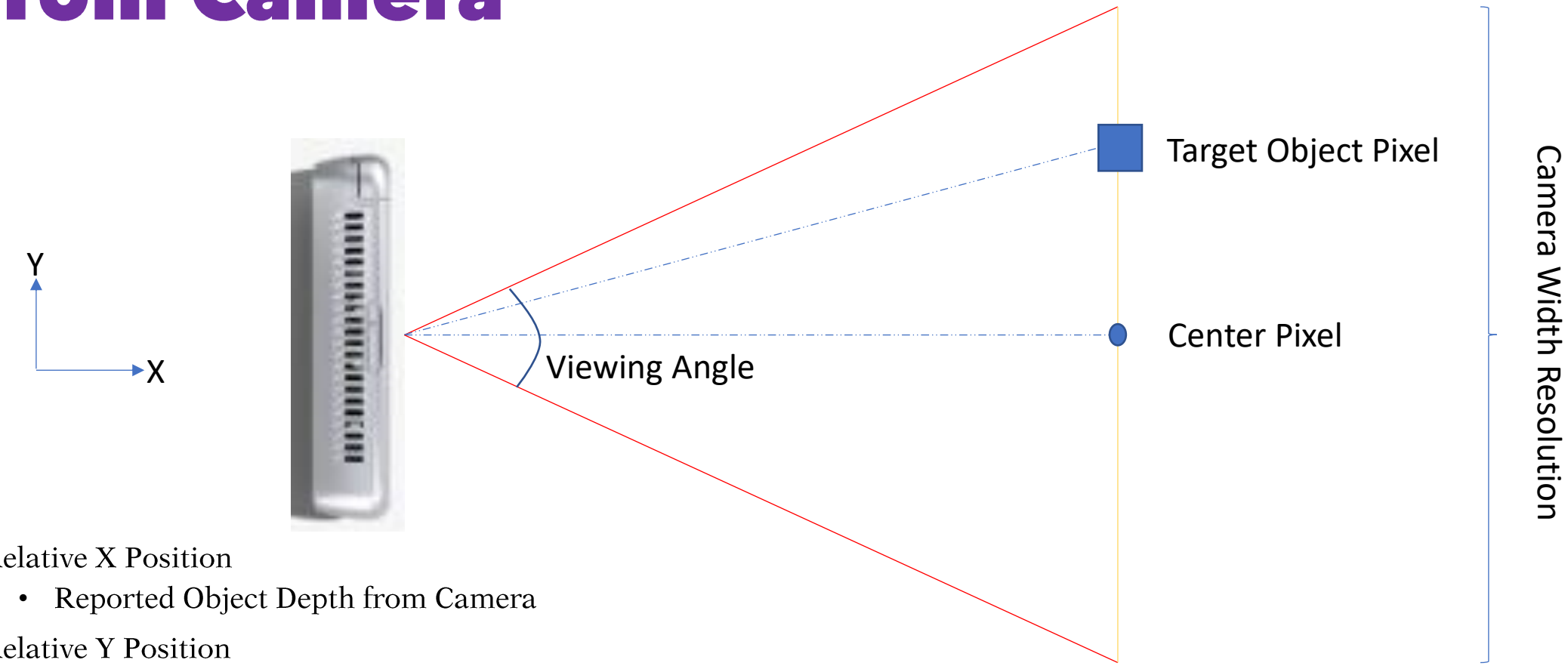
- SLAM continues to run while navigating
  - Dynamic obstacle avoidance and map updating
- Robot performs continuous re-calculation of desired position and heading
  - In case an obstacle requires a change of position and heading

Transforming Relative Arm and Camera Positioning to Global Frame

- Get current robot position
- Get current joint angles
- Calculate global gripper position based off relative position
  - X: global robot X position
    + cos(relative heading)*relative X position
    + cos(relative heading+pi/2)*relative Y position
  - Y: global robot Y position
    + sin(relative heading)*relative X position
    + sin(relative heading+pi/2)*relative Y position
- Calculate distance and heading between current position and desired position on global frame
- Create arm path using compute_cartesian_path
- Set planned path as arm goal



Object Position Triangulation from Camera

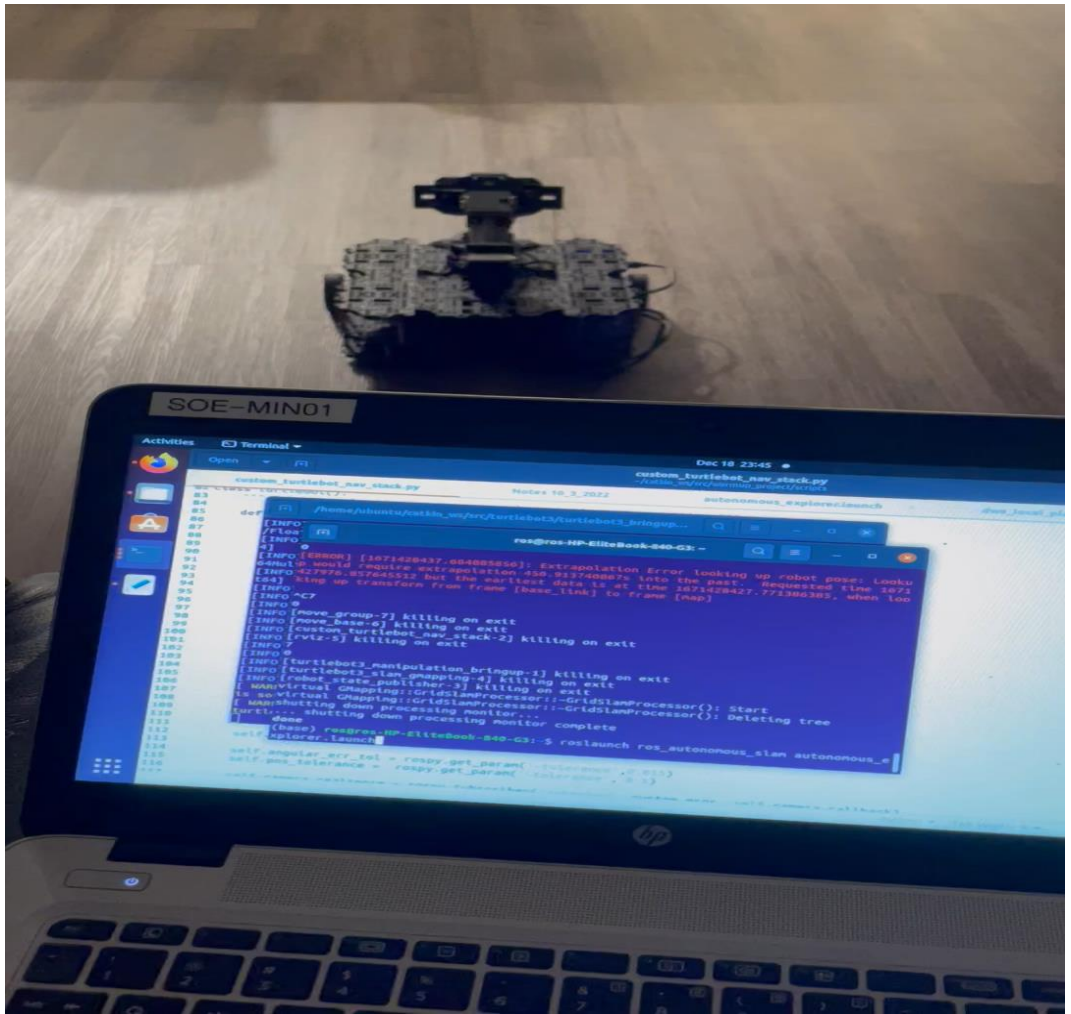


- Relative X Position
  - Reported Object Depth from Camera
- Relative Y Position
  - tan((width resolution/2 – center x pixel)/width resolution*viewing angle)*object depth
- Relative Z Position
  - tan((height resolution/2 – center y pixel)/height resolution*viewing angle)*object depth

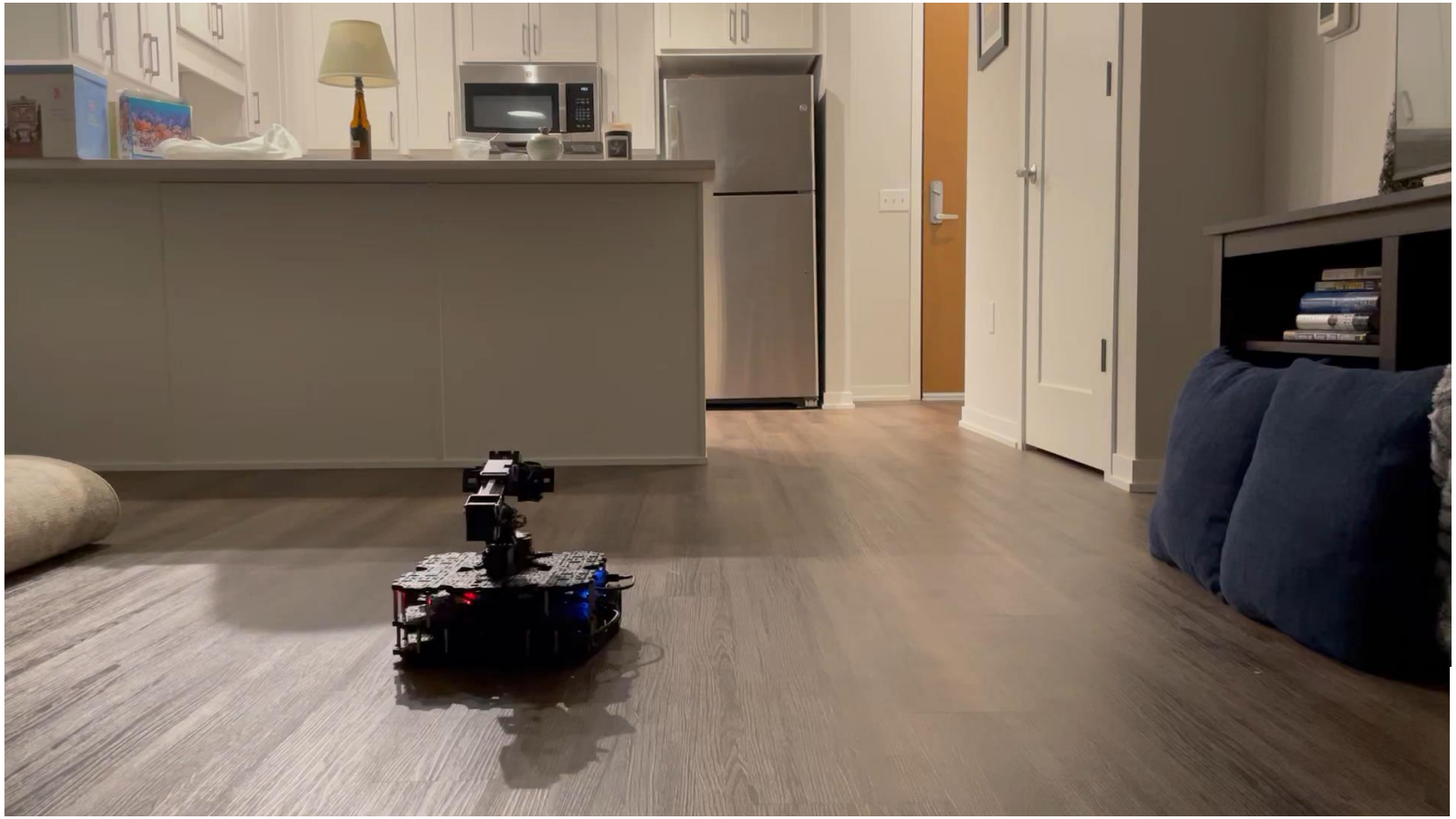
Demos

School of Engineering | UNIVERSITY OF St.Thomas

Video Demonstrating SLAM Mapping and Autonomous Exploration

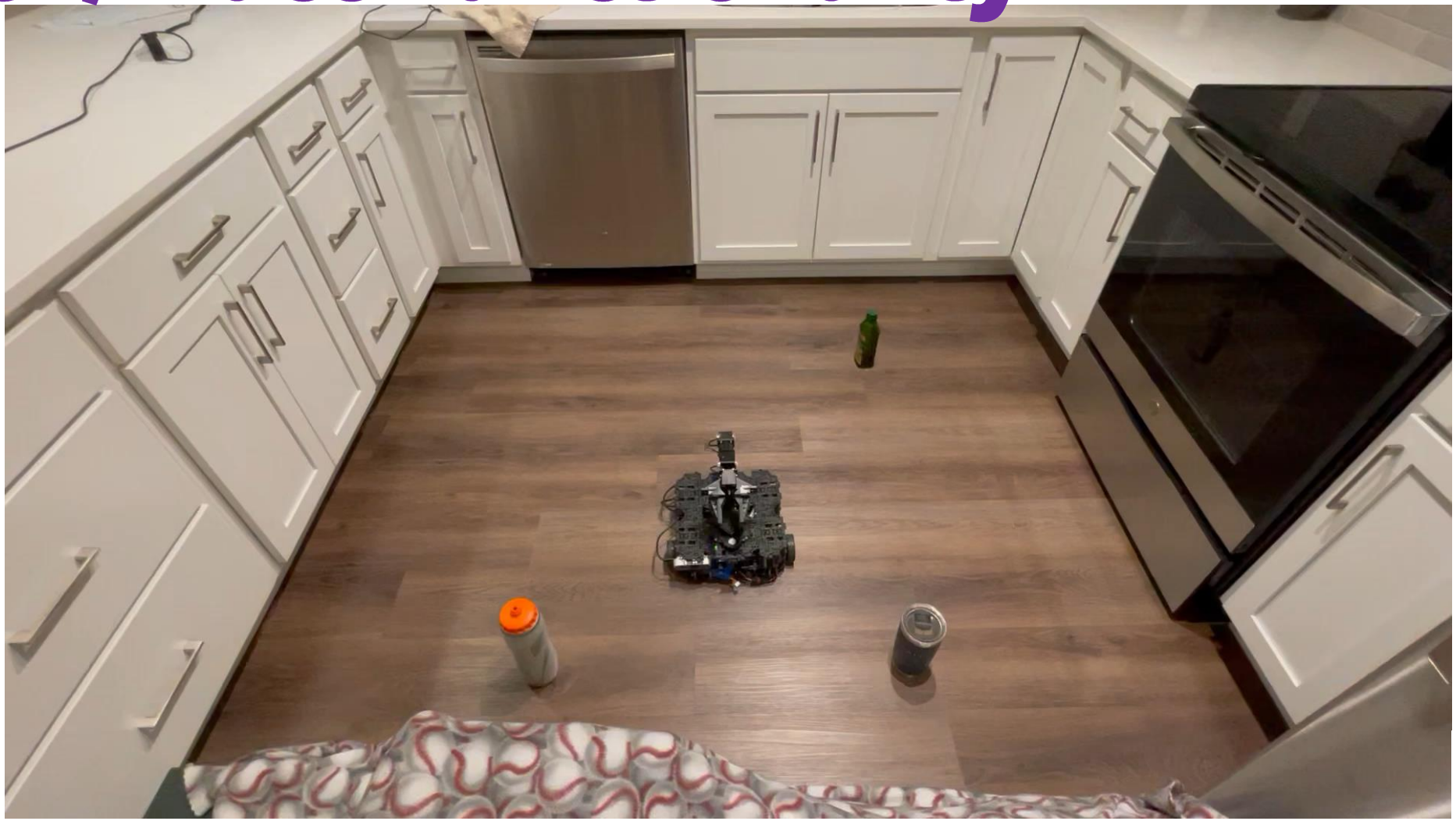


Video Demonstrating Dynamic Obstacle Avoidance

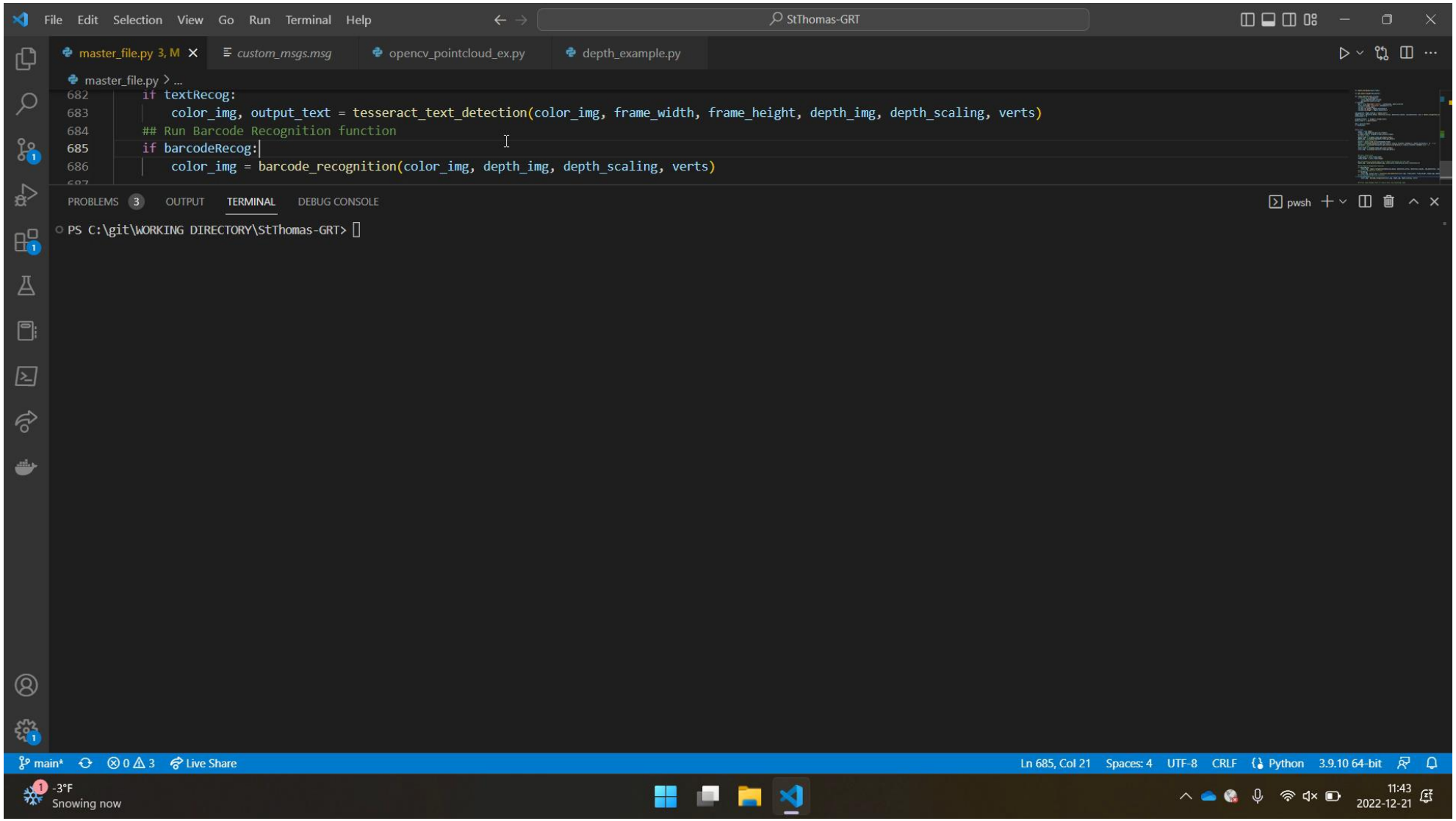


Video Demonstrating Pick/Place Functionality

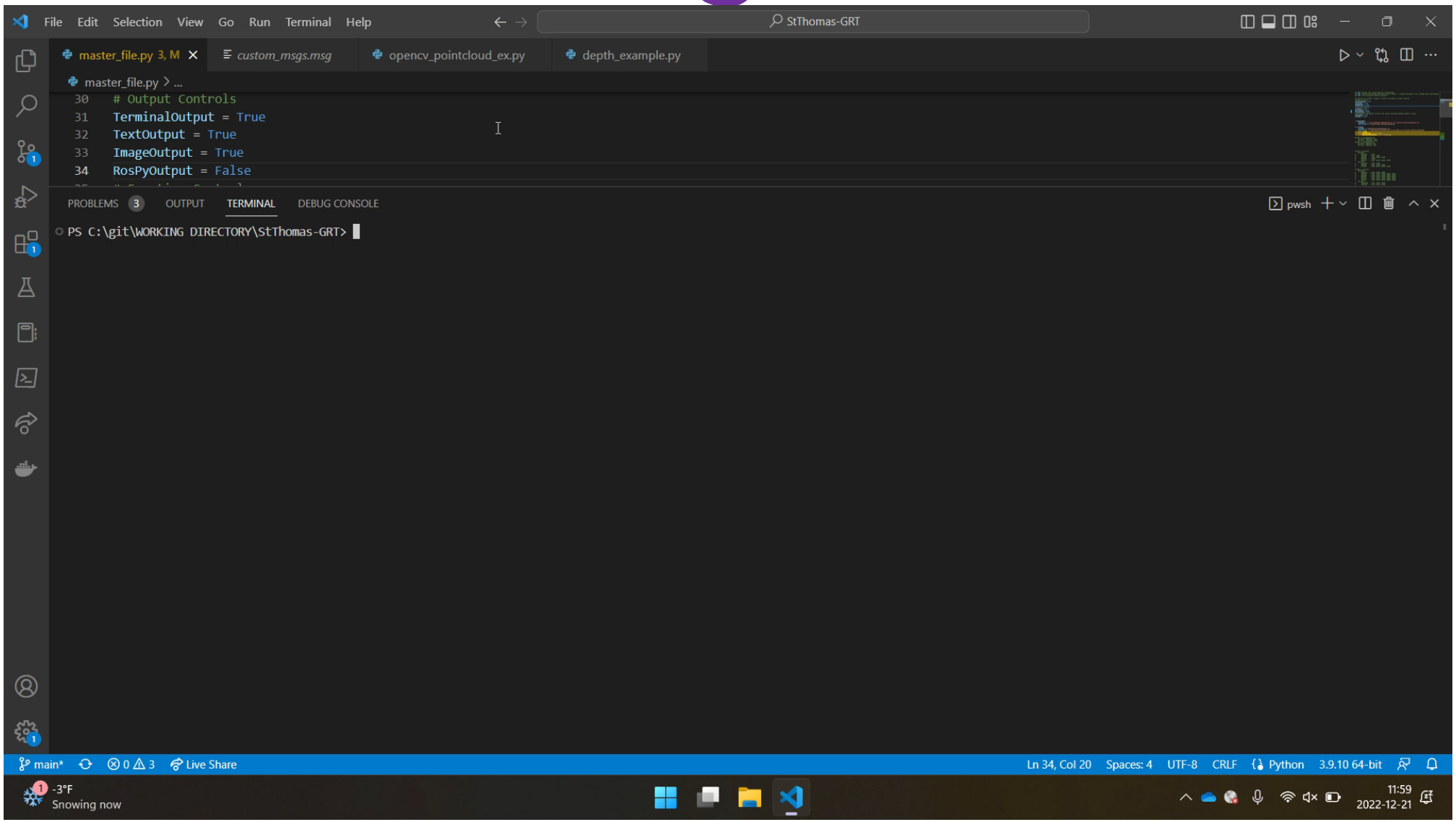
School of Engineering | UNIVERSITY OF St.Thomas



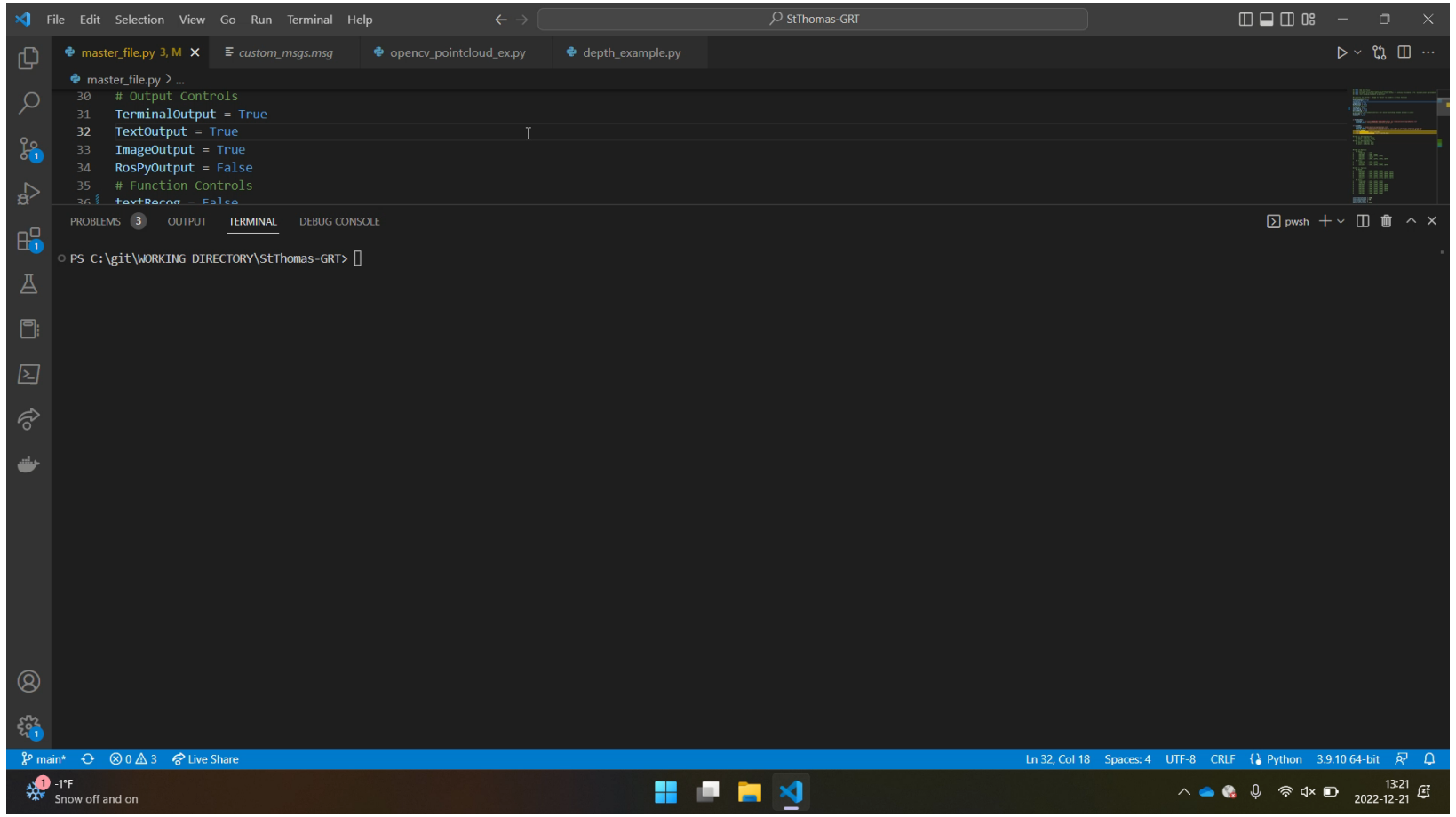
Demo - Object Recognition



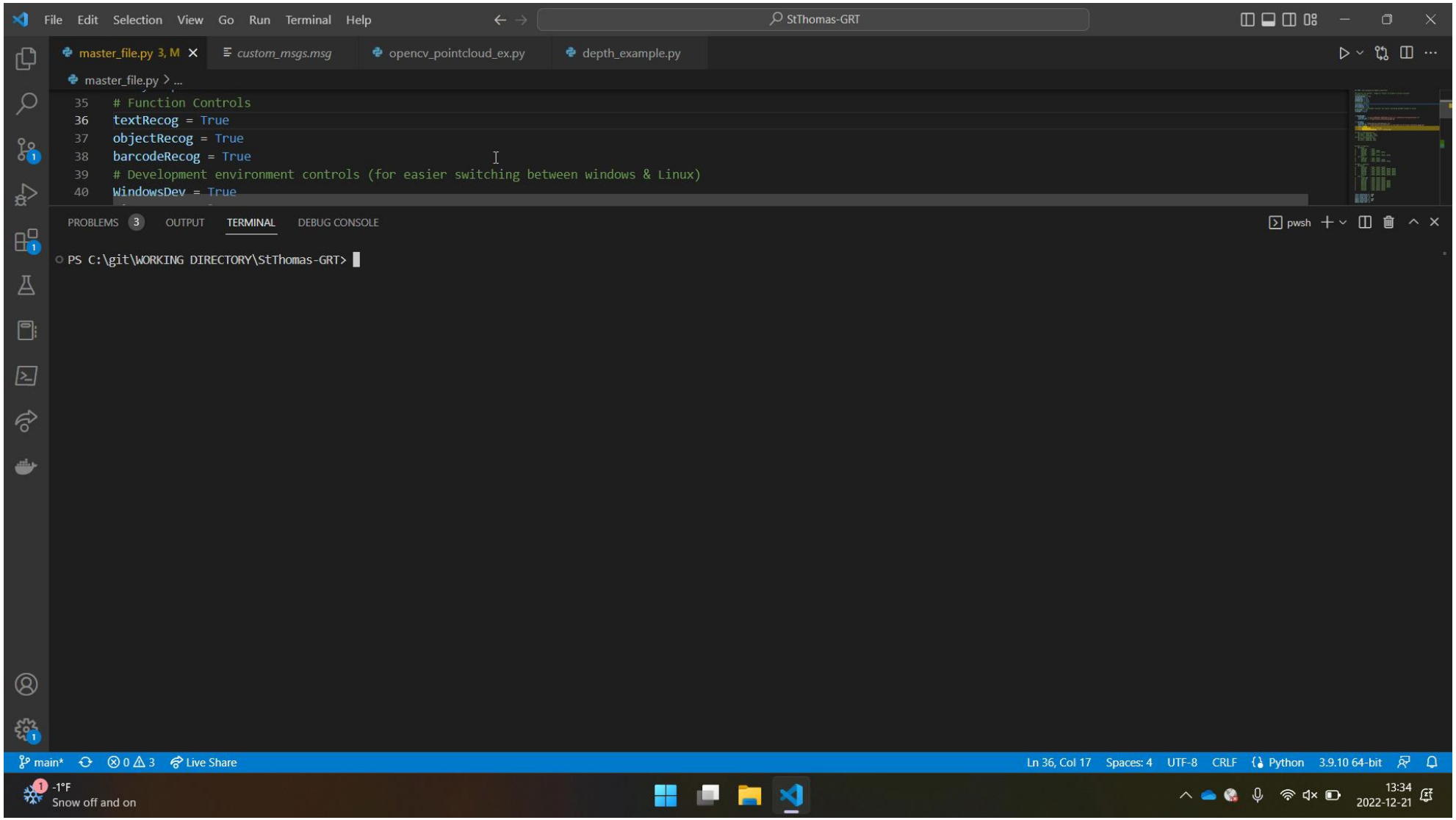
Demo - Text Recognition



Demo - Barcode Recognition



Demo - Combined Recognition



Future Improvements

Future Improvements

- Upgrade to Raspberry Pi 5
  - Faster speeds
  - More memory
  - Upgrade would improve streaming rate and allow for smoother navigation
- Algorithm Improvements
  - Improve object coordinate algorithm
  - Where to grab an object as opposed to the body center
- Securely mount the camera to the robot
- Integrating Raspberry Pi Camera or Depth Camera into SLAM

Camera Specific Improvements

- Point Cloud accuracy improvements
- Implement an auto-focus that is object-, text-, or barcode-specific
- Optimize resolution & frame rates for Raspberry Pi hardware
- Change input format from USB camera connection to streamed ROS data connection
  - Be wary of network bandwidth
- Exploration of other camera stream options & formats
- Depth error contextualization
- Auto color correction

Recognition Specific Improvements

School of Engineering | UNIVERSITY OF St.Thomas

- Text Recognition
  - Word grouping and recognition
  - Simple goal: group recognized letters by size and proximity
  - Larger goal: recognize words
    - Combination of recognized letters (two models: letters & words)
    - Model output with a word (single model: words)
  - Frame to Frame Averaging
- Object Recognition
  - Train an application specific model
- Barcode Recognition
  - Connection to internet or other database for contextualization of the observed barcode