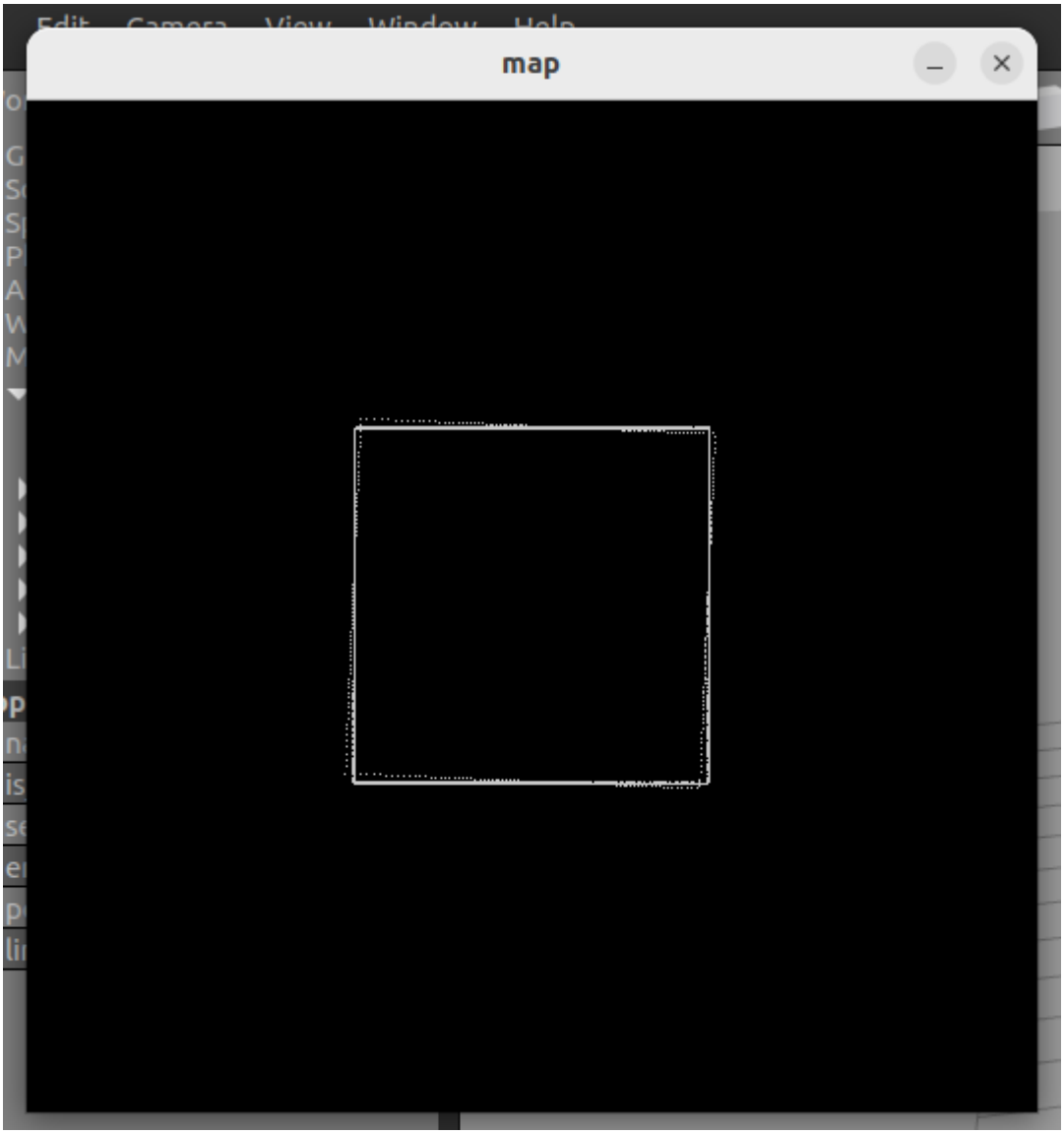
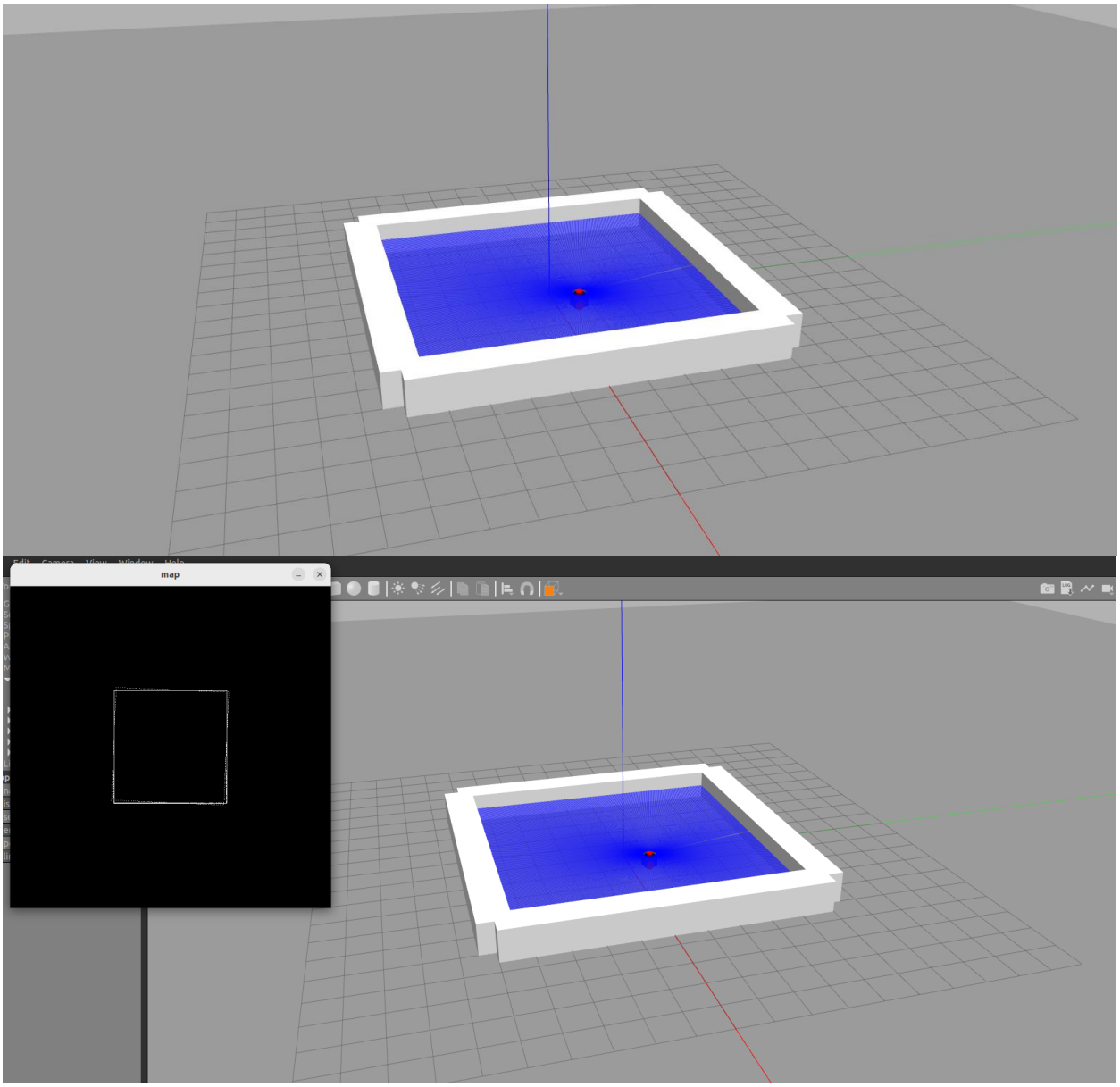
Christopher Budd
218919068


map


map

Adjustments to add_obstacle

```
# A very simple ros node to populate the world
#
import rclpy
import os
from gazebo_msgs.srv import SpawnEntity, DeleteEntity

def make_obstacle(node, id, x0, y0, h, w, l):
   CYLINDER_MODEL = """
       <sdf version="1.6">        \
         <world name="default">                          \
           <model name="obstacle">       \
             <static>true</static>       \
             <link name="all">                           \
               <collision name="one">     \
                 <pose>0 0 {o} 0 0 0</pose>       \
                 <geometry>       \
                   <cylinder>                          \
                     <radius>{r}</radius>              \
                     <length>{h}</length>              \
                   </cylinder>        \
                  </geometry>       \
               </collision>       \
               <visual name="two">      \
                 <pose>0 0 {o} 0 0 0</pose>       \
                 <geometry>       \
                   <cylinder>                          \
                     <radius>{r}</radius>              \
                     <length>{h}</length>              \
                   </cylinder>                         \
                 </geometry>        \
               </visual>        \
             </link>                                     \
           </model>         \
         </world>                                        \
       </sdf>"""

   BOX_MODEL = """
        <sdf version="1.6">                \
          <world name="default">            \
            <model name="obstacle">         \
              <static>true</static>         \
              <link name="all">              \
                <collision name="one">       \
```

```
                <pose>0 0 {o} 0 0 0</pose>  \
                <geometry>                  \
                  <box>                     \
                    <size>{l} {w} {h}</size> \
                  </box>                    \
                </geometry>                 \
              </collision>                  \
              <visual name="two">           \
                <pose>0 0 {o} 0 0 0</pose>  \
                <geometry>                  \
                  <box>                     \
                    <size>{l} {w} {h}</size> \
                  </box>                    \
                </geometry>                 \
              </visual>                     \
            </link>                          \
          </model>                           \
        </world>                             \
      </sdf>"""

    client = node.create_client(SpawnEntity, "/spawn_entity")
    node.get_logger().info("Connecting to /spawn_entity service...")
    client.wait_for_service()
    node.get_logger().info("...connected")
    request = SpawnEntity.Request()
    request.name = id
    request.initial_pose.position.x = float(x0)
    request.initial_pose.position.y = float(y0)
    request.initial_pose.position.z = float(0)
    dict = {'h' : h, 'l':l, 'w':w, 'o': h/2}
    request.xml = BOX_MODEL.format(**dict)
    node.get_logger().info(f"Making request...")
    future = client.call_async(request)
    while not future.done():
        rclpy.spin_once(node)
    node.get_logger().info("...done")
    if not future.result().success:
        node.get_logger().info(f"Failure {future.result()}")

def remove_obstacle(node, id):
    client = node.create_client(DeleteEntity, "/delete_entity")
    node.get_logger().info("Connecting to /delete_entity service...")
    client.wait_for_service()
    node.get_logger().info("...connected")
    request = DeleteEntity.Request()
```

```
    request.name = id
    node.get_logger().info("Making request...")
    future = client.call_async(request)
    rclpy.spin_until_future_complete(node, future)
    node.get_logger().info("...done")
    if not future.result().success:
        node.get_logger().info(f"Failure {future.result()}")



def main(args=None):
  rclpy.init(args=args)
  node = rclpy.create_node('demo')
  make_obstacle(node, 'blob', 2, 3, 2, 1)

  node.destroy_node()
  rclpy.shutdown()

if __name__ == '__main__':
    main()
```

build_map to create the map

```
# Populate the world with a json map file

#
import sys
import json
import rclpy
from ament_index_python.packages import get_package_share_directory
from .add_obstacle import make_obstacle

def main(args=None):
    rclpy.init(args=args)
    node = rclpy.create_node('build_map')

    node.declare_parameter('map', 'default.yaml')
    map_name = node.get_parameter('map').get_parameter_value().string_value
    package_path = get_package_share_directory('cpmr_ch2')

    try:
        with open(f"{package_path}/{map_name}") as fd:
            map = json.load(fd)
    except Exception as e:
        node.get_logger().error(f"Unable to find/parse map in
{package_path}/{map_name}")
```

```
        sys.exit(1)

    for o in map.keys():
        node.get_logger().info(f"Populating map with {map[o]}")
        make_obstacle(node, o, map[o]['x'], map[o]['y'], 1.0, map[o]['w'],
map[o]['l'])
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

json file

```
{
  "o1" : {"x" : -5, "y" : 0, "l" : 1, "w" : 10},
  "o2" : {"x" : 5, "y" : 0, "l" : 1, "w" : 10},
  "o3" : {"x" : 0, "y" : -5, "l" : 10, "w" : 1},
  "o4" : {"x" : 0, "y" : 5, "l" : 10, "w" : 1}
}
```

Lidar map generation

```
class CollectLidar(Node):
    _WIDTH = 513
    _HEIGHT = 513
    _M_PER_PIXEL = 0.05



    cur_x = 0
    cur_y = 0
    cur_t = 0

    def __init__(self):
        super().__init__('collect_lidar')
        self.get_logger().info(f'{self.get_name()} created')

        self._map = np.zeros((CollectLidar._HEIGHT, CollectLidar._WIDTH),
dtype=np.uint8)
        self.get_logger().info(f"map init {self._map}")
```

```
        self.create_subscription(Odometry, "/odom", self._odom_callback, 1)
        self.create_subscription(LaserScan, "/scan", self._scan_callback, 1)



    def _scan_callback(self, msg):
        angle_min = msg.angle_min
        angle_max = msg.angle_max
        angle_increment = msg.angle_increment
        ranges = msg.ranges
        # self.get_logger().info(f"Ranges ({ranges})\n\n")
        self.get_logger().info(f"lidar
({angle_min},{angle_max},{angle_increment},{len(ranges)})")

        for r in enumerate(ranges):
            # self.get_logger().info(f"Range {r}")
            scan_t = self.cur_t + angle_min + r[0] * angle_increment
            if r[1] < 10:
                self.get_logger().info(f"Range {r}")
                row = int(int(self._HEIGHT/2) - (self.cur_y + r[1] *
math.sin(scan_t)) / self._M_PER_PIXEL)
                col = int(int(self._WIDTH/2) + (self.cur_x + r[1] *
math.cos(scan_t)) / self._M_PER_PIXEL)
                self.get_logger().info(f"Map {row} {col}")
                if (row >= 0) and (col >= 0) and (row < self._HEIGHT) and (col <
self._WIDTH):
                    self._map[row, col] = 250

        cv2.imshow('map',self._map)
        cv2.waitKey(10)

    def _odom_callback(self, msg):
        pose = msg.pose.pose

        self.cur_x = pose.position.x
        self.cur_y = pose.position.y
        o = pose.orientation
        roll, pitchc, yaw = euler_from_quaternion(o)
        self.cur_t = yaw

        self.get_logger().info(f"at ({self.cur_x},{self.cur_y},{self.cur_t})")

def main(args=None):
    rclpy.init(args=args)
```

```
    node = CollectLidar()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Questions

1. How did you decide to represent the map and update it when sensor measurements are obtained?
   I represent the map in a 2D array that was located in lidar collector/ collect_lidar
   I updated based on how close the ping is to prevent distanced reading, and the position of the robot assuming the odom is 100% accurate. Once the x and y of the point is calculated, I fill in that spot in the array. I then display the map.

2. How did you decide where to drive the robot?

   I did it based on the object and testing the map. After setting up the box, I added in a few other objects to the map. One key movement was driving all the way around an object to get the full object in the map.

3. Was the model consistent over all measurement obtained?

   No, there were some points that were added that were not from an object. This was due to the robot design where it sometimes wobbles and the lidar gets a ping from the floor. There were some other issues with going around objects but likely an error with my math.