HW3: Rocky Times Challenge - Search, Map and Analyze

RAS 598: Space Robotics and AI

Aldrin Inbaraj A
ASU ID: 1226200393

1 Introduction

The objective of this assignment was to design a robust drone system that can autonomously search for and identify cylindrical rock formations of varying heights in a simulated lunar terrain, estimate their dimensions, and perform a precision landing on the tallest structure.

2 Setting Up the Environment

The simulation environment for this assignment was set up using the PX4 SITL, Gazebo, and ROS 2 Humble ecosystem. The core codebase for the assignment was cloned from the course repository into the ROS 2 workspace:

```
cd ~/ros2_ws/src
git clone https://github.com/inbarajaldrin/RAS-SES-598-Space-Robotics-and-AI.git
cd ~/ros2_ws/src
ln -s RAS-SES-598-Space-Robotics-and-AI/assignments/terrain_mapping_drone_control .
```

PX4-Autopilot was installed separately, and the simulation was compiled using a custom drone model named `gz_x500_depth_mono`:

```
cd ~/PX4-Autopilot
make px4_sitl gz_x500_depth_mono
```

To ensure compatibility between PX4 and the drone model, a helper script was used to deploy the necessary SDF and configuration files into the PX4 workspace:

```
cd ~/ros2_ws/src/terrain_mapping_drone_control
chmod +x scripts/deploy_px4_model.sh
./scripts/deploy_px4_model.sh -p ~/PX4-Autopilot
```

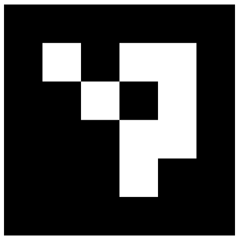
To establish communication between PX4 and the ROS 2 nodes via DDS, the Micro XRCE Agent was launched:

```
MicroXRCEAgent udp4 -p 8888
```

Next, QGroundControl was started to monitor the drone's status and validate PX4 SITL communication:

```
./QGroundControl-x86_64.AppImage
```

Initially, both the tall and short cylinders had the same ArUco marker ID (0), which made them indistinguishable to the drone. To resolve this, the marker on the shorter cylinder was replaced with ArUco ID 1, enabling the drone to differentiate between them during detection.


(a) ArUco Marker ID 0 (Tall Cylinder)

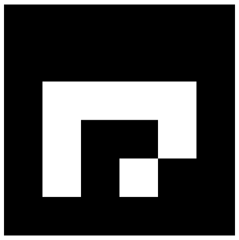

(b) ArUco Marker ID 1 (Short Cylinder)

Figure 1: Updated ArUco markers used to distinguish tall and short cylinders.

Finally, the simulation was launched using the provided ROS 2 launch file, which initialized the Gazebo environment, spawned the drone and cylinder models, and brought up all relevant ROS 2 nodes.

```
ros2 launch terrain_mapping_drone_control cylinder_landing.launch.py
```

With the gazebo world launched and the ros_gz_bridges running, we launch

```
ros2 launch terrain_mapping_drone_control mission.launch.py
```

This runs the aruco_tracker.py and auto_detect_land.py codes to perform the mission and the performance is logged in the terminal upon completion of the mission.

3 Autonomous Mission Flow

The core mission logic for the drone is implemented in a custom ROS 2 node named `CylinderMission`, defined in `auto_detect_land.py`. The node operates as a state machine, handling drone control, perception, and decision-making through a series of well-defined states.

The following outlines each mission state and its role in the overall landing pipeline:

- **WAIT_INTRINSICS**: Waits until camera intrinsics and battery status are received. Once available, logs the starting battery percentage and transitions to `ARM_TAKEOFF`.
- **ARM_TAKEOFF**: Executes a two-stage takeoff:
  1. Ascends vertically to $(0, 0, -5)$.
  2. Moves horizontally to the circle entry point at $(15, 0, -5)$.

  On reaching the entry point, the drone calculates the starting angle and transitions into circular flight.
- **CIRCLE**: Flies in a circular trajectory around the origin at a fixed radius of 15 meters and altitude of $z = -5$. Position targets are computed using:

  ```
  x = radius * math.cos(theta)
  y = radius * math.sin(theta)
  z = altitude
  self.theta += circle_speed
  ```

  RGBD data is used in parallel to detect cylindrical structures.
- **SERVO**: Once a cylinder is detected, the drone adjusts its position to maintain a distance of 15 meters from the object using depth measurements. If detection fails within 5 seconds, it reverts to the `CIRCLE` state.

- **HOVER**: Holds position for 7 seconds while accumulating bounding box measurements from the RGBD data. Median height and width values are computed and used to distinguish cylinders. If the object is new, its dimensions are stored and the drone resumes circling. If it's a previously seen cylinder, the drone proceeds to landing.
- **ARUCO_HOVER**: Descends to $z = -20$ to get a stable view of ArUco markers placed atop both cylinders.
- **ARUCO_SELECT**: Selects the ArUco marker with the smallest Z value (i.e., the tallest cylinder in the scene) as the landing target. Marker positions are parsed and transformed from ArUco to drone frame using:

  ```
  drone_x = aruco_y
  drone_y = aruco_x
  drone_z = aruco_z
  ```

- **ARUCO_MOVE**: Navigates to the selected marker's position. Once the drone is within 0.5 meters, it initiates the landing sequence.
- **ARUCO_LAND**: Sends PX4 the `VEHICLE_CMD_NAV_LAND` command and disarms upon touchdown.
- **COMPLETE**: Logs mission time and battery usage (if available), then transitions to `DONE`, ending the mission.

This mission flow enables the drone to autonomously take off, explore the environment, detect and evaluate rock formations, and perform precision landing on the tallest cylinder using ArUco markers for final guidance.

4 Cylinder Detection Pipeline

Cylindrical object detection is performed using synchronized RGB and depth images from the front-facing camera. The pipeline runs continuously during the mission and is triggered within the `CIRCLE` state to identify potential rock formations in the scene.

The image callback processes synchronized RGB and depth frames using the following steps:

1. **Color Segmentation:** The RGB image is converted to HSV, and a mask is generated using thresholding to isolate the cylinder color:

   ```
   hsv = cv2.cvtColor(rgb, cv2.COLOR_BGR2HSV)
   lower_hsv = np.array([0, 0, 110])
   upper_hsv = np.array([180, 40, 180])
   color_mask = cv2.inRange(hsv, lower_hsv, upper_hsv) > 0
   ```

2. **Depth Filtering:** A depth mask filters out points outside the expected cylinder range (1m to 30m), and the final mask is the intersection of depth and color:

   ```
   depth_mask = np.logical_and(depth > 1.0, depth < 20.0)
   object_mask = np.logical_and(depth_mask, color_mask)
   ```

3. **Morphological Processing:** A morphological closing operation is applied to reduce noise:

   ```
   object_mask = cv2.morphologyEx(object_mask.astype(np.uint8),
                                  cv2.MORPH_CLOSE, np.ones((5, 5), np.uint8))
   ```

4. **Contour Detection:** The pipeline extracts contours with sufficient area to locate bounding boxes around objects of interest:

   ```
   contours , _ = cv2.findContours(object_mask , ...)
   filtered = [c for c in contours if cv2.contourArea(c) > min_pixel_area]
   ```

5. **Measurement Estimation:** For the largest valid bounding box, the median depth is extracted from the ROI. The object's width and height are estimated using pinhole projection:

   ```
   Z = float(np.median(roi))
   width_m = (w * Z) / self.fx
   height_m = (h * Z) / self.fy
   ```

6. **State Transition:** If a valid measurement is detected while circling, the system transitions to the `SERVO` state to initiate closer inspection and prepare for dimensional analysis.

This detection approach uses both depth and color information to achieve robust segmentation in the presence of shadows. While it performs reliably in simulation, lighting variations and depth sensor noise introduce a measurement error margin of approximately ±0.5 meters.

5 Performance Metrics

The mission was evaluated based on total duration, energy usage, and the accuracy of cylinder detection and landing.

- **Total Mission Duration:** 267 seconds
- **Battery Consumed:** 0%
  (Battery data remained static throughout the mission via `/fmu/out/battery_status`.)

Cylinder Detection Accuracy

Using synchronized RGB and depth data, the system estimated the following cylinder dimensions:

- **Short Cylinder:** Detected ≈ **1.11m** × **7.02m** (Ground Truth: 1.0m × 7.0m)
- **Tall Cylinder:** Detected ≈ **0.98m** × **10.06m** (Ground Truth: 1.0m × 10.0m)

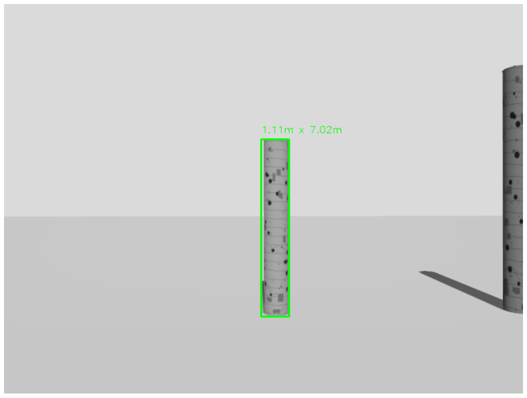


Figure 2: Short Cylinder Detection

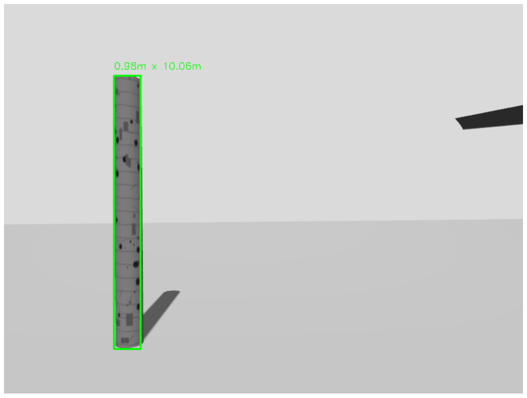


Figure 3: Tall Cylinder Detection

Figure 4: Cylinder detection results with dimension overlays

Landing Outcome

A full demonstration of the mission — including autonomous takeoff, cylinder detection, ArUco tracking, and successful landing — is available at this **link**.

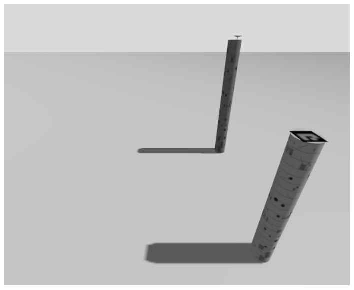


Figure 5: Drone landed on the taller cylinder at mission completion

6 Conclusion

This assignment involved building a complete autonomous drone pipeline using PX4 SITL, ROS 2, and Gazebo to detect, analyze, and land on cylindrical rock formations. The mission flow included synchronized RGBD perception, circular exploration, bounding box–based measurement, ArUco marker localization, and final precision landing.

Setting up the environment itself was a significant challenge. Getting the correct versions of `ros_gz_bridge` and Gazebo (eventually settling on `harmonic`) took considerable effort. Once configured, the system was able to run in simulation and complete the mission reliably.

There are a few limitations in the current implementation:

- Some behaviors, such as the circle radius and starting altitude, are hardcoded.
- Cylinder measurement has a tolerance of approximately ± 0.5 meters, largely due to shadow artifacts in the RGB image.
- Battery consumption tracking could not be verified as the voltage-based SOC field remained constant throughout the mission.

Despite these issues, the drone successfully completed its mission by autonomously selecting the taller cylinder and performing a controlled landing.

7 Future Work

Several improvements can be made to extend the current system and address its limitations:

- **Better performance logging:** Due to incomplete battery status feedback from PX4 SITL, a workaround is needed to measure energy usage accurately.
- **Remove hardcoded logic using SLAM:** Instead of relying on predefined positions and circle flight, future implementations can use RTAB-Map to build a 3D map of the environment and dynamically generate exploration and landing trajectories.
- **Fusion of RGB and depth data:** Incorporating both modalities for segmentation and measurement would improve precision and reduce the impact of lighting variations or shadows.
- **Model-agnostic detection:** A more generalized object detection pipeline could enable the drone to detect and analyze rocks or structures of arbitrary shape and size.

With additional time and hardware-in-the-loop testing, this system could be extended to more complex multi-agent lunar exploration scenarios.