Team Members & Sponsors

Peter: Software
Swetha: LiDAR, mapping, Jetson
Cher: Drone, QGroundControl
Tiffany: LiDAR, mapping, logo
Jackie: Controller Interface, MAVLink

Phil Tokumaru - Project Advisor
Scott Rasmussen - Drone Specialist
List of Parts

- Varmint Quad Drone
- NVIDIA Jetson TX2
- TI mmWave Radar Sensor
- Slamtec RPLiDAR Sensor
Block Diagram

Managing Process
- Joystick
- USB on Laptop
- Validator: Python Process
- Drone Link

Telemetry Radio

Router

PX4
- MAVLink

Jetson
- RADAR
- LiDAR
while True:

    action = controller.get_action()

    3d_data = await drone.get_sensor_data()

    safe_action = validator.propose_action(action, 3d_data)

    drone.send_joystick(safe_action)
Control Loop Flow

while True:

    action = controller.get_action()

    3d_data = await drone.get_sensor_data()

    safe_action = validator.propose_action(action, 3d_data)

    drone.send_joystick(safe_action)
RADAR & LiDAR package

- Drone has Jetson TX2 onboard which merges the RADAR and LiDAR data
- Server waits for drone sensor data and treats it as one data source
- RADAR and LiDAR sensors are supported by custom backend drivers

Get data from sensors → Apply spatial transformations to data → ‘Pickle’ data to binary, and send to server
TI mmWave RADAR

- Custom drivers to get data from the sensors, manage sensor failures, and automatically update data orientation.
- Data arrives at ~10Hz over WiFi
LiDAR & Hector SLAM

- Simultaneous Localization And Mapping
- LiDAR creates a 2D map of current surroundings
- Hector SLAM package creates a continuous map while drone is in motion
  - Scan Matching: compares last scan with current scan to find minimum error between scan & map
  - Pose Estimates: using fast update rate & consecutive scans to determine where drone is relative to map, replacing need for odometry
Hector SLAM

1. Initial Scan
2. Map
3. Scan Again
4. Estimate Pose Change of New Scan
5. Align Scan to Pose Estimate
6. Update Map
Hector SLAM

Initial Scan -> Map -> Scan Again

Map

Estimate Pose Change of New Scan -> Align Scan to Pose Estimate -> Update Map
Hector SLAM

Initial Scan → Map → Scan Again

→ Estimate Pose Change of New Scan

Align Scan to Pose Estimate → Update Map
Hector SLAM

1. Initial Scan
2. Map
3. Scan Again
   - Estimate Pose Change of New Scan
   - Align Scan to Pose Estimate
   - Update Map
Hector SLAM

1. Initial Scan
2. Map
3. Scan Again
4. Estimate Pose Change of New Scan
5. Align Scan to Pose Estimate
6. Update Map
Hector SLAM

1. Initial Scan
2. Map
3. Scan Again
4. Estimate Pose Change of New Scan
5. Align Scan to Pose Estimate
6. Update Map
Hector SLAM

1. Initial Scan
2. Map
3. Scan Again
4. Estimate Pose Change of New Scan
5. Align Scan to Pose Estimate
6. Update Map
Hector SLAM Data

- Pose information: x, y, z, w
- Use x and y for location
- Use z and w for rotation onto proper axis

```
x: 0.127863751221, y: 0.299377441406, z: 0.361930816041, w: 0.9322804958365
x: 0.27534484863, y: 0.42894744873, z: 0.321432049791, w: 0.846935781652
x: 0.277534484863, y: 0.42894744873, z: 0.321432049791, w: 0.846935781652
x: 0.330375671387, y: 0.517162522998, z: 0.30365518356, w: 0.952781994738
x: 0.330375671387, y: 0.517162522998, z: 0.30365518356, w: 0.952781994738
x: 0.2214317627, y: 0.735667147217, z: 0.318599649426, w: 0.95846823835
x: 0.2214317627, y: 0.735667147217, z: 0.318599649426, w: 0.95846823835
x: 0.32137298584, y: 0.801277156645, z: 0.312675952828, w: 0.94959857307
x: 0.32137298584, y: 0.801277156645, z: 0.312675952828, w: 0.94959857307
x: -0.015443481445, y: 0.875682830811, z: 0.386613453234, w: 0.922441854276
x: -0.015443481445, y: 0.875682830811, z: 0.386613453234, w: 0.922441854276
x: -0.10882305254, y: 0.870086669922, z: 0.395646886645, w: 0.918403693182
x: -0.10882305254, y: 0.870086669922, z: 0.395646886645, w: 0.918403693182
x: -0.099739874707, y: 0.986174560547, z: 0.40669316435, w: 0.913842278651
x: -0.099739874707, y: 0.986174560547, z: 0.40669316435, w: 0.913842278651
x: -0.113498687744, y: 1.01376131592, z: 0.392499829536, w: 0.91752418759
x: -0.113498687744, y: 1.01376131592, z: 0.392499829536, w: 0.91752418759
x: -0.113067626953, y: 1.02454376221, z: 0.373605169932, w: 0.92758727108
x: -0.113067626953, y: 1.02454376221, z: 0.373605169932, w: 0.92758727108
x: -0.099479675293, y: 1.02656121826, z: 0.360847436553, w: 0.932624858951
x: -0.099479675293, y: 1.02656121826, z: 0.360847436553, w: 0.932624858951
x: -0.12317276001, y: 1.01394599888, z: 0.381755585799, w: 0.92344376064
```
Control Loop Flow

```python
while True:
    action = controller.get_action()
    3d_data = await drone.get_sensor_data()
    safe_action = validator.propose_action(action, 3d_data)
    drone.send_joystick(safe_action)
```
We estimate the drone position in two parts:

1. Estimate the location the drone moved between the data sample and now
2. Add a vector based on the new action proposed by the user

\[
\text{estimation} = (\text{current} + \text{time\_passed} \times \text{drone\_velocity}) + \text{new\_action\_vector}
\]
while True:

    action = controller.get_action()

    3d_data = await drone.get_sensor_data()

    safe_action = validator.propose_action(action, 3d_data)

    drone.send_joystick(safe_action)
MAVLink Joystick Control

Traditional Control System:
- RC Transmitter
  - FrSky Taranis X9 Lite S
- QGroundControl
MAVLink Joystick Control

Our Control System:

- **Joystick Controller**
  - pygame library
  - 4 axis outputs \([-1:1]\)
- Sending Mavlink commands via telemetry radio
  - Holybro Transceiver Telemetry Radio
    - 915 MHz
MAVLink Joystick Control

MAVLink - [mavlink.io/en/](http://mavlink.io/en/)

- Lightweight messaging protocol
- Compatible with various autopilots (flight controllers) and ground stations
  - PX4, ArduPilot, iNab, etc.
  - QGroundControl, Mission Planner, MAVProxy, UgCS, etc.
- Messages defined with XML files
- Variety of MAVLink APIs that provide implementation
  - pymavlink

### MAVLink Packet Format

```
uint8_t magic; ///< protocol magic marker
uint8_t len;  ///< Length of payload
uint8_t seq;  ///< Sequence of packet
uint8_t sysid;///< ID of message sender system/aircraft
uint8_t compid;///< ID of the message sender component
uint8_t msgid 0:7;///< first 8 bits of the ID of the message
uint8_t msgid 8:15;///< middle 8 bits of the ID of the message
uint8_t msgid 16:23;///< last 8 bits of the ID of the message
uint8_t payload[max 255];///< A maximum of 255 payload bytes
uint16_t checksum;///< CRC-16/MCRF4XX
```

### MAVLink v1 Frame (8 - 263 bytes)

```
<table>
<thead>
<tr>
<th>STX</th>
<th>LEN</th>
<th>SEQ</th>
<th>SYS ID</th>
<th>COMP ID</th>
<th>MSG ID</th>
<th>PAYLOAD (0 - 255 bytes)</th>
<th>CHECKSUM (2 bytes)</th>
</tr>
</thead>
</table>
```
MAVLink Joystick Control

- MAVLink Message (#400): MAV_CMD_COMPONENT_ARM_DISARM
  - Arms and disarms the drone

```
# Arm
# master.arducopter_arm() or:
master.mav.command_long_send(
    master.target_system,
    master.target_component,
    mavutil.mavlink.MAV_CMD_COMPONENT_ARM_DISARM,
    0,  # confirmation
    1,  # arm:1/disarm:0
    0, 0, 0, 0, 0)
```

- MAVLink Message (#69): MANUAL_CONTROL
  - API for manually controlling vehicle using standard joystick axes nomenclature

```
# Manual Control
buttons = 1 + 1 << 3 + 1 << 7
master.mav.command_long_send(
    master.target_system,
    master.target_component,
    mavutil.mavlink.MANUAL_CONTROL,
    0,  # confirmation
    0,  # ROLL
    0,  # PITCH
    500, # THROTTLE
    0,  # YAW
    buttons)
```
Recap

```python
while True:
    action = controller.get_action()
    3d_data = await drone.get_sensor_data()
    safe_action = validator.propose_action(action, 3d_data)
    drone.send_joystick(safe_action)
```
Drone & RC

How it works

1. Taranis lite access x9 transmitter connects to R-XSR receiver on the drone
2. Arm motors
3. Flying in Mode two
   a. Left Joystick: Throttle
   b. Right Joystick: Directional

Struggles

- Motor placement was backwards
- Drifting
- Difficulty landing
- Lost hardware
Timeline

Fall 2020

- Configured QGroundControl
- 2D real-time LiDAR map
- Collision detection with RADAR

Winter 2021

- Drone operates properly
- Continuous LiDAR map
- Map joystick inputs to proper MAVLink commands

Spring 2021

- Install sensors on drone
- Consolidate all the individual working parts
Acknowledgments

Yogananda Isukapalli - Capstone Instructor
Boning Dong - TA
Trenton Rochelle - TA
Phil Tokumaru - Project Advisor
Scott Rasmussen - Drone Specialist
Questions?