Flying Penguins

Embedded Linux applications for autonomous UAVs
Clay McClure

github.com/claymation
autopilot
RC input
motor mixing
stabilization
 telemetry
missions
failsafes
AUTO PILOT ≠ AUTO NOMOUS
“system finds its own goal positions”

where to go
how to get there
what to do next
SO MANY ALGORITHMS, SO LITTLE COMPUTER
Autopilot *runs on* Linux
Autopilot *talks to* Linux
ODROID-XU3 Lite

- Samsung Exynos5422 **octa** core
  - 4x Cortex™-A15 2.0GHz
  - 4x Cortex™-A7 1.4GHz
- 2 GB RAM
- 32+ GB flash
- 4x USB 2.0 + 1x USB 3.0
Roadmap (so far)

UAV

? 

Your App

Linux
Roadmap (so far)

UAV  MAV Link  Your App

Linux
MAVLink is the HTTP of drones
(it’s also the libcurl)
status
configuration
position / attitude
setpoints
missions
Roadmap (so far)

UAV → MAV Link → Your App

Linux
Roadmap (so far)

- UAV
- MAV Link
- Middleware
- Your App
- Linux
Middleware

DroneAPI

- Python

- Go to Kevin Hester’s talk tomorrow

mavros

- Python, C++, Lisp (really)

- Access to a wealth of robotics research and tools
Roadmap (so far)

UAV → MAV Link → mavros → Your App

Linux ?
Roadmap (so far)

UAV

MAV Link

mavros

Your App

ROS

Linux
ROS Crash Course
Robot Operating System

“ROS is an open-source, meta-operating system for your robot.”
ROS is an open-source, meta-operating system for your robot.
Topics

Node

Topic

Node

Node

Node
Services

Node

Node

↓  ↑

↓  ↑
but wait, that’s not all…

parameters
dynamic reconfig
coordinate frames
transformations
record/playback
visualization
logging
Roadmap

UAV → MAV Link → mavros → Your App

ROS

Linux
mavros is the Babel fish of drones
Topics

/mavros/state
/mavros/imu/data
/mavros/global_position/global
/mavros/local_position/local
/mavros/setpoint_position/local_position
/mavros/setpoint_velocity/cmd_vel
Services

/mavros/cmd/arming
/mavros/cmd/land
/mavros/cmd/takeoff
/mavros/set_mode
/mavros/set_stream_rate
PX4 + ROS

DEEPLY EMBEDDED CONTROLLER
- Actuator & Sensor Interface
- Geofence & Safety Controller
- uORB pub/sub bus
- Trajectory Control
- Attitude Control

LINUX COMPANION COMPUTER
- SLAM
- Obstacle Avoidance
- ROS pub/sub bus
- Attitude Control
- Trajectory Control

> 250 Hz
< 1 ms latency

swappable execution location

Credit: Kabir Mohammed
Roadmap

UAV → MAV Link → mavros → Your App

ROS

Linux
YAPL

Yet Another Precision Lander
Event-driven programming

• “Don’t call me, I’ll call you”

• Your application code responds to events
  
  • Message arrival
    
    • “my position is (x, y, z)”
  
  • Timer expiry
    
    • “it’s time to run the control loop”
Nodes

• **Tracker**
  - Processes video stream, looks for landing pad
  - Publishes target position/velocity messages

• **Commander**
  - Subscribes to vehicle state and position messages
  - Subscribes to target tracker messages
  - Controls vehicle velocity
class TrackerNode(object):

    def __init__(self):
        rospy.init_node("tracker")

        use_sim = rospy.get_param("~use_sim", False)
        camera_matrix = rospy.get_param("~camera_matrix")

        # ...

        self.image_publisher = \
            rospy.Publisher("tracker/image", 
                sensor_msgs.msg.Image, queue_size=1)

        self.track_publisher = \
            rospy.Publisher("tracker/track", 
                Track, queue_size=1)
class TrackerNode(object):
    
    def __init__(self):
        rospy.init_node("tracker")

        use_sim = rospy.get_param("~/use_sim", False)
        camera_matrix = rospy.get_param("~/camera_matrix")

        # ...

        self.image_publisher = \
            rospy.Publisher("tracker/image", \
                sensor_msgs.msg.Image, queue_size=1)

    self.track_publisher = \
        rospy.Publisher("tracker/track", \
            Track, queue_size=1)
$ cat msg/Track.msg
# Whether we're tracking an object
std_msgs/Bool is_tracking

# Relative position and velocity of the tracked object
geometry_msgs/Vector3 position
geometry_msgs/Vector3 velocity
def publish_track(self, position, velocity):
    msg = TrackStamped()
    msg.track.is_tracking.data = self.is_tracking

    if self.is_tracking:
        msg.track.position.x = position[0]
        msg.track.position.y = position[1]
        msg.track.position.z = position[2]
        msg.track.velocity.x = velocity[0]
        msg.track.velocity.y = velocity[1]
        msg.track.velocity.z = velocity[2]

    self.track_publisher.publish(msg)
def publish_image(self, image):
    msg = self.image_bridge.cv2_to_imgmsg(image, "bgr8")
    self.image_publisher.publish(msg)
simulation
HITL

• Hardware in the loop

• Flight software runs on flight hardware

• Simulated sensor and control inputs
SITL

- Software in the loop
- Flight software runs on (Linux) desktop
- Simulated sensor and control inputs and HAL
“In theory there is no difference between theory and practice. In practice there there is is.”

~ Yogi Berra
Practical Considerations
Connections

• UART recommended
  • Requires 6-pin DF-13, possibly a level shifter

• USB works for me
  • Use hot glue gun

• sudo apt-get remove modemmanager
Power

5V 5A UBEC

ODROID
+ USB camera
+ WiFi
+ 3S LiPo
= 5 hours
Launch files

- ROS feature that makes it easy to start and manage multiple nodes and their parameters

- roslaunch lander lander.launch
Startup

• Use ubuntu’s upstart to launch ROS + mavros + application nodes

• rosr
  run robot_upstart install \lander/launch/lander.launch
Telemetry

- MAVLink + 3DR radio
- WiFi
  - Ad-Hoc mode (`man wireless`)
  - Need high-gain antenna and a tracker (helper)
  - `sudo apt-get remove wpasupplicant`
- GSM?
Coordinate Frames

- Global / Local
  - NED
  - ENU
- Body-fixed
- tf library
In closing...
What will you make?
For more information...

ros.org
ardupilot.com
pixhawk.org/start
pixhawk.ethz.ch/mavlink
github.com/mavlink/mavros
github.com/claymation/lander