Autonomous Landing on a Moving Platform

presented by
Ross Kidson, Karol Hausman, Sebastian Nagel
Motivation

- Controlled landing in difficult environments
- Following the marker implicitly
- Control with the Android Smartphone
State Graph

- Manual Control
- Estimate the Position (EKF)
- Find the Marker
- Landing on the Marker (PID)
- Following the Marker (PID)
- Control Command from the mobile
Two parts of the project

- Project was split into two tasks:
  - Android interface – Sebastian Nagel
  - Controlling the ARDrone – Ross Kidson, Karol Hausman
Android Interface

- User Interface for the mobile phone
Android Interface - Technologies

- **Rosjava** – ROS client library purely in Java

- **Android_core** – package developed by Google to integrate Rosjava clients in Android projects
Main differences to roscpp:

- Asynchronous:
  - because of heavy dependence on network communication
- More threads, fewer processes
  - No spin() method!
  - Many nodes in one JVM process – similar to nodelets
- Use in memory communication
Android Interface - Android_core

- Android_core libraries:
  - Classes to use Rosjava in Android applications (Master chooser, RosActivity, …)
  - Tutorial projects and sample implementations of ROS nodes in Android Views (e.g. RosImageView)
Android Interface - Implementing

- **ARDroneRemote node**
  - Publisher/Subscriber for communication with controller:
    ```cpp
    // State publisher, autonomous availability subscriber
    private Publisher<std_msgs.Int32> pubState;
    private Subscriber<std_msgs.Bool> subAvailable;
    ```
  - Publisher/Subscriber for communication with ARDrone driver:
    ```cpp
    // Manual control publishers
    private Publisher<std_msgs.Empty> pubTakeOff, pubLand, pubEmergency;
    private Publisher<geometry_msgs.Twist> pubVel;
    ```
Example code

Creating new publisher in onStart() of NodeMain:

```java
@Override
public void onStart(ConnectedNode node) {
    // State control
    pubState = node.newPublisher("/visnav/state", std_msgs.Int32._TYPE);
    [...]
}
```

Publishing a message:

```java
public void setState(State state) {
    Int32 msg = pubState.newMessage();
    msg.setData(state.ordinal());
    pubState.publish(msg);
    [...]
}
```
Autonomous Part - Plan

- Use Kalman Filter for localization of the ARDrone (detecting the first marker)
- Try to detect another marker and set it as the goal for the PD controller
- Follow the detected moving marker
- Land on the moving marker if the quadrocopter is stable enough
Two different marker following approaches:

- Use one marker for both EKF positioning and moving marker tracking
- Use one marker for tracking and another one for EKF positioning
The same two approaches

Procedure for ensuring marker tracking stability

Height controller → reduce height while landing

If the quadrocopter is too low to see the marker → stabilize the position → land
Both approaches were compared

- Distance from landed position to marker used as metric
- Static landing pad in this case
- 15 Trials for both cases

<table>
<thead>
<tr>
<th></th>
<th>Beta marker (landing marker not used for localization)</th>
<th>Zeta marker (landing marker used for localization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ave. dist</td>
<td>19.7cm</td>
<td>34.5cm</td>
</tr>
<tr>
<td>Ave. dist (exl. Misses)</td>
<td>11.1cm</td>
<td>18.1cm</td>
</tr>
<tr>
<td>st. dev (exl. Misses)</td>
<td>6.3cm</td>
<td>10.9cm</td>
</tr>
<tr>
<td># misses</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Ave. time taken</td>
<td>31.14s</td>
<td>34.08s</td>
</tr>
</tbody>
</table>
Autonomous Part - Result

- Video 1
Combining two parts

- Architecture

- Problems
Final Result

- Video 2
Conclusions

Possible improvements
- More stable controller
- Transfer the software to the cell phone
- Motion model
- Markerless tracking/landing

Applications
- Autonomous re-charging
- Explore and return home

Experience gained during the project
Questions?

Thank you for your attention.