Instituto Nacional de Astrofísica,
Óptica y Electrónica.

Combinación de un Controlador PID y el Sistema Slam para Micro-Vehículos Aéreos.

Presenta:
Dr. José Martínez Carranza.
Aldrich Alfredo Cabrera Ponce.
Roberto Munguia Silva.

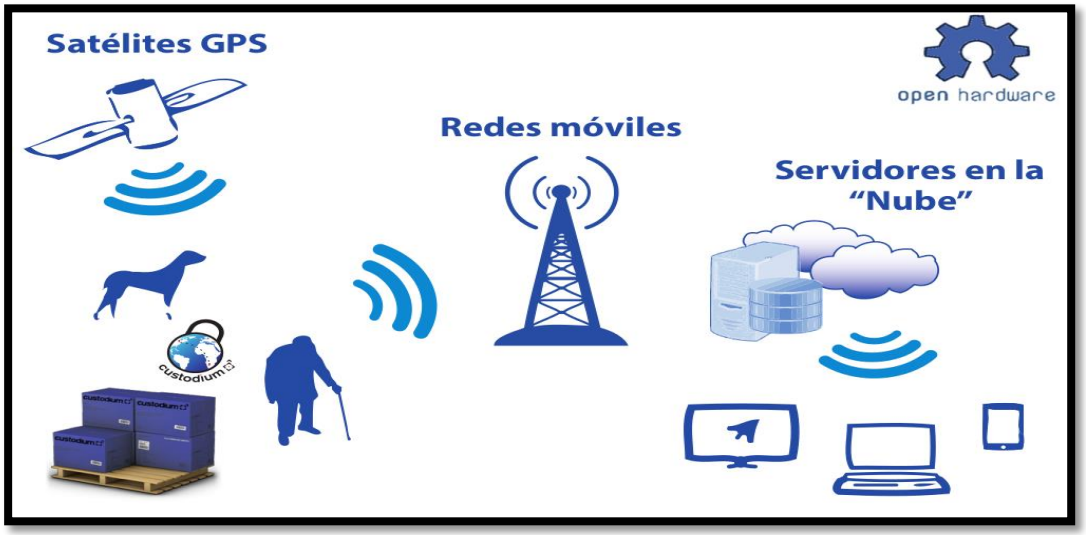
Contenido

- Introducción
- Objetivos
- Drone (Bebop)
- ORB-SLAM2
- PID (Proporcional – Integral - Derivativo)
- Vuelo Semi-Autónomo

Introducción

La navegación para la robótica aérea abarca prácticas, técnicas y procedimientos que permiten conducir una aeronave a su lugar de destino. El piloto debe tener los conocimientos necesarios para asegurar la integridad de vehículo aéreo.

Hoy en día, la navegación se usa con mayor frecuencia en sistemas robóticos en el cual se implementan en sistemas SLAM para localización mediante la visión. Las áreas que utilizan estos sistemas pueden ser robótica móvil, robótica doméstica entre otras.



2. Objetivos

Objetivos particulares:

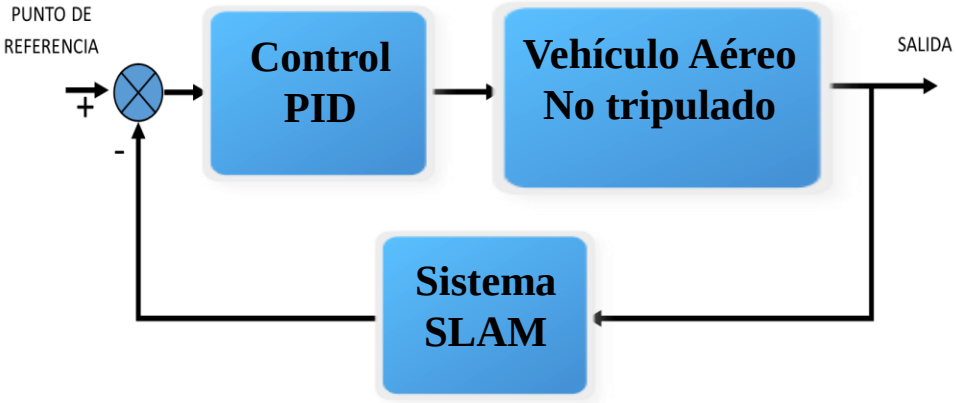
1.Controlar un vehículo aéreo mediante la computadora.
2.Diseñar un control PID para navegación autónoma del vehículo.

Objetivo

Desarrollar un control de vuelo semi-autónomo para un vehículo aéreo no tripulado basado en localización por visión.

Drone Parrot

Bebop Drone



Ubuntu y ROS.

LTS 14.04 - *Indigo*

3. Drone Bebop

El vehículo aéreo con el que se trabajara será un bebop de la marca Parrot. Este vehículo usa una cámara optical: "Fisheye" 186° 1/2,3" 920x1080p (30fps) lo cual nos ayudara con la localización del mismo. Para ello es necesario realizar los siguiente pasos:

* Crear un espacio de trabajo a través de ROS para nuestro vehículo aéreo.

* Acceder y visualizar la cámara del vehículo en nuestras computadoras.

* Programar nuestro vehículo en el lenguaje de programación de Python.

* Pilotear el vehículo mediante el Keyboard de la computadora.

Crear el work_space

Para crear nuestro espacio de trabajo se debe realizar lo siguiente:

Instalar librerias de catkin_tools:

```
$ sudo apt-get install build-essential python-rosdep python-catkin-tools
```

```
# Create and initialize the workspace
$ mkdir -p ~/bebop_ws/src && cd ~/bebop_ws
$ catkin init
$ git clone https://github.com/AutonomyLab/bebop_autonomy.git src/bebop_autonomy
# Update rosdep database and install dependencies
$ rosdep update
$ rosdep install --from-paths src -i
# Build the workspace
$ catkin build -DCMAKE_BUILD_TYPE=RelWithDebInfo
```

Cambiar por Catkin_make

Visualización de la cámara

Source devel/setup.bash

Se realiza un source esto es para asegurar de que su área de trabajo se superponga correctamente por el script de instalación. Sin embargo, se puede poner a través de la terminal el siguiente comando.

Sudo gedit ~/.bashrc

Al entrar al bashrc al final del documento ponemos la siguiente línea:

source ~/bebop_ws/devel/setup.bash

Visualización de la cámara

Una vez que se aseguro el espacio de trabajo se pasa a visualizar el lo que observa nuestro vehículo aéreo en nuestra computadora:

```
$ roslaunch bebop_driver bebop_node.launch
```

bebop_node.launch¶

```
<?xml version="1.0"?>
<launch>
    <arg name="namespace" default="bebop" />
    <arg name="ip" default="192.168.42.1" />
    <arg name="config_file" default="$(find bebop_driver)/config/defaults.yaml" />
    <arg name="camera_info_url" default="package://bebop_driver/data/bebop_camera_calib.yaml" />
    <group ns="$(arg namespace)">
        <node pkg="bebop_driver" name="bebop_driver" type="bebop_driver_node" output="screen">
            <param name="camera_info_url" value="$(arg camera_info_url)" />
            <param name="bebop_ip" value="$(arg ip)" />
            <rosparam command="load" file="$(arg config_file)" />
        </node>
        <include file="$(find bebop_description)/launch/description.launch" />
    </group>
</launch>
```

Visualización de la cámara

```
$ roslaunch bebop_tools bebop_nodelet_iv.launch
```

bebop_tools/launch/bebop_nodelet_iv.launch¶

```
<?xml version="1.0"?>
<launch>
    <!-- include the nodelet launch file from bebop_driver -->
    <arg name="namespace" default="bebop" />
    <include file="$(find bebop_driver)/launch/bebop_nodelet.launch">
      <arg name="namespace" value="$(arg namespace)" />
    </include>
    <!-- use the same nodelet manager and namespace, then load image_view nodelet -->
    <group ns="$(arg namespace)">
       <node pkg="nodelet" type="nodelet" name="bebop_image_view_nodelet"
          args="load image_view/image bebop_nodelet_manager">
          <remap from="image" to="image_raw" />
        </node>
    </group>
</launch>
```

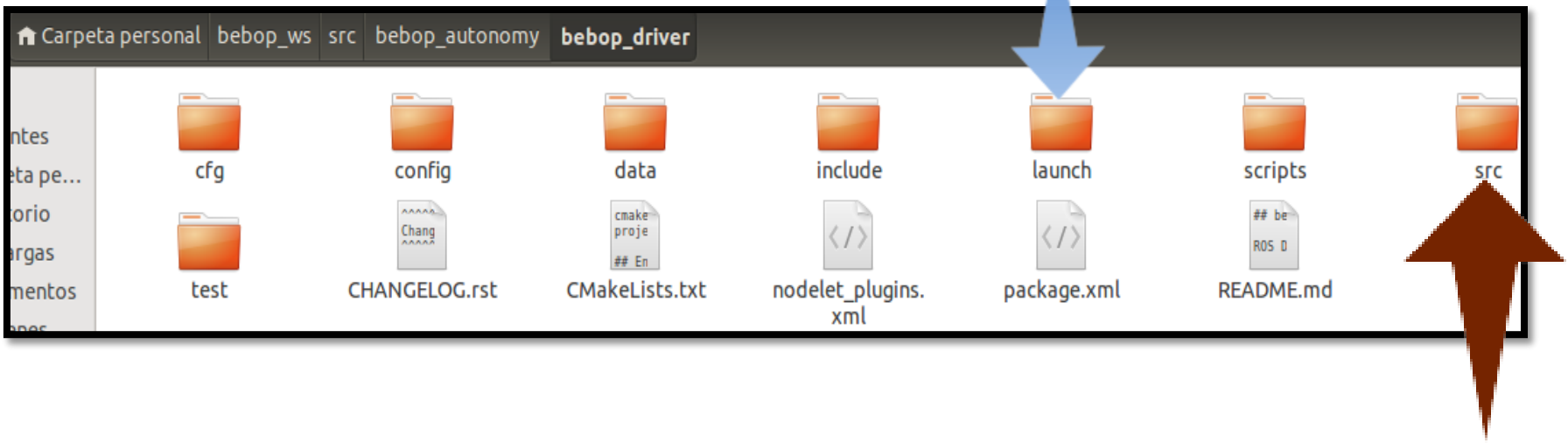
Visualización de la cámara

bebop_driver/launch/bebop_nodelet.launch¶

```
<?xml version="1.0"?>
<launch>
    <arg name="namespace" default="bebop" />
    <arg name="ip" default="192.168.42.1" />
    <arg name="config_file" default="$(find bebop_driver)/config/defaults.yaml" />
    <arg name="camera_info_url" default="package://bebop_driver/data/bebop_camera_calib.yaml" />
    <group ns="$(arg namespace)">
        <!-- nodelet manager -->
        <node pkg="nodelet" type="nodelet" name="bebop_nodelet_manager" args="manager" output="screen"/>
        <!-- bebop_nodelet -->
        <node pkg="nodelet" type="nodelet" name="bebop_nodelet"
          args="load bebop_driver/BebopDriverNodelet bebop_nodelet_manager">
            <param name="camera_info_url" value="$(arg camera_info_url)" />
            <param name="bebop_ip" value="$(arg ip)" />
            <rosparam command="load" file="$(arg config_file)" />
        </node>
        <include file="$(find bebop_description)/launch/description.launch" />
    </group>
</launch>
```

Programación del vehículo

Carpeta donde estarán los lanzadores.



Códigos para mandar comandos al drone.

Programación del vehículo 1

Dentro de la carpeta src ponemos nuestro primer código lo cual será manipular el vehículo aéreo con el keyboard.

keyboard.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import rospy
from geometry_msgs.msg import Twist
from std_msgs.msg import Int8
from std_msgs.msg import Empty

# Finally the GUI libraries
from PySide import QtCore, QtGui

GUI_UPDATE_PERIOD = 100 #ms

# Here we define the keyboard map for our controller (note that python has no enums, so we use a class)
class KeyMapping(object):
    PitchForward     = QtCore.Qt.Key.Key_W
    PitchBackward    = QtCore.Qt.Key.Key_S
    RollLeft         = QtCore.Qt.Key.Key_A
    RollRight        = QtCore.Qt.Key.Key_D
    YawLeft          = QtCore.Qt.Key.Key_Q
    YawRight         = QtCore.Qt.Key.Key_E
    IncreaseAltitude = QtCore.Qt.Key.Key_Up
    DecreaseAltitude = QtCore.Qt.Key.Key_Down
    Takeoff          = QtCore.Qt.Key.Key_T
    Land             = QtCore.Qt.Key.Key_Space
    Hovering         = QtCore.Qt.Key.Key_H
    Release          = QtCore.Qt.Key.Key_R

    CamUp           = QtCore.Qt.Key.Key_I
    CamDown         = QtCore.Qt.Key.Key_K
    CamRigth        = QtCore.Qt.Key.Key_J
    CamLeft         = QtCore.Qt.Key.Key_L

class MainWindow(QtGui.QMainWindow):
    def __init__(self):
        super(MainWindow,self).__init__()
        self.setWindowTitle('Bebop interfaz')
```

Programación del vehículo 2

*keyboard.py

```
36          self.setWindowTitle('Bebop interfaz')
37          self.redrawTimer = QtCore.QTimer(self)
38          self.redrawTimer.timeout.connect(self.update)
39          self.redrawTimer.start(GUI_UPDATE_PERIOD)
40
41          self.pitch = 0
42          self.roll = 0
43          self.yaw_velocity = 0
44          self.tilt = 0
45          self.cam_pan = 0
46          self.z_velocity = 0
47          self.arm = False
48          self.toff = False
49          self.ld = False
50          self.override = False
51
52          self.pubCommandP = rospy.Publisher('/keyboard/cmd_vel',Twist,queue_size=10)
53          self.pubLand    = rospy.Publisher('/keyboard/land',Int8,queue_size = 10)
54          self.pubTakeoff = rospy.Publisher('/keyboard/takeoff',Int8,queue_size = 10)
55          self.pubOverride = rospy.Publisher('/keyboard/override',Int8,queue_size = 10)
56          self.keyb = Twist()
57
58      # We add a keyboard handler to the DroneVideoDisplay to react to keypresses
59      def keyPressEvent(self, event):
60          self.override = 1
61          self.pubOverride.publish(self.override)
62          key = event.key()
63          # If we have constructed the drone controller and the key is not generated from an auto-repeating key
64          #if not event.isAutoRepeat():
65          if not event.isAutoRepeat():
66              # Handle the important cases first!
67              if key == KeyMapping.Hovering:
68                  self.keyb.linear.x  = 0.0
69                  self.keyb.linear.y  = 0.0
70                  self.keyb.linear.z  = 0.0
71                  self.keyb.angular.z = 0.0
```

Programación del vehículo 3

*keyboard.py

```
71                  self.keyb.angular.z = 0.0
72                  self.pubCommandP.publish(self.keyb)
73                  rospy.loginfo("Hovering")
74
75              elif key == KeyMapping.Takeoff:
76                  self.toff = not self.toff
77                  if self.toff == True:
78                      #self.pubTakeoff.publish(Empty())
79                      self.pubTakeoff.publish(1)
80                      self.toff = False
81                  rospy.loginfo("TAKING OFF")
82
83              elif key == KeyMapping.Land:
84                  self.ld = not self.ld
85                  if self.ld == True:
86                      self.pubLand.publish(1)
87                      self.ld = False
88                  rospy.loginfo("LANDING")
89
90              else:
91                  # Now we handle moving, notice that this section is the opposite (+=) of the keyrelease section
92                  if key == KeyMapping.YawLeft:
93                      self.yaw_velocity += 0.3
94                      rospy.loginfo("YAW_LEFT")
95
96                  elif key == KeyMapping.YawRight:
97                      self.yaw_velocity += -0.3
98                      rospy.loginfo("YAW_RIGHT")
99
100                 elif key == KeyMapping.PitchForward:
101                     self.pitch += 0.3
102                     rospy.loginfo("FORWARD")
103
104                 elif key == KeyMapping.PitchBackward:
105                     self.pitch += -0.3
106                     rospy.loginfo("BACKWARD")
```

Programación del vehículo 4

*keyboard.py

```
106                     rospy.loginfo("BACKWARD")
107
108                 elif key == KeyMapping.RollLeft:
109                     self.roll += 0.3
110                     rospy.loginfo("LEFT")
111
112                 elif key == KeyMapping.RollRight:
113                     self.roll += -0.3
114                     rospy.loginfo("RIGTH")
115
116                 elif key == KeyMapping.IncreaseAltitude:
117                     self.z_velocity += 0.3
118                     rospy.loginfo("UP")
119
120                 elif key == KeyMapping.DecreaseAltitude:
121                     self.z_velocity += -0.3
122                     rospy.loginfo("DOWN")
123
124                 elif key == KeyMapping.CamUp:
125                     self.tilt += 5
126                 elif key == KeyMapping.CamDown:
127                     self.tilt += -5
128                 elif key == KeyMapping.CamLeft:
129                     self.cam_pan += 5
130                 elif key == KeyMapping.CamRigth:
131                     self.cam_pan += -5
132
133
134             self.keyb.linear.x  = self.pitch
135             self.keyb.linear.y  = self.roll
136             self.keyb.linear.z  = self.z_velocity
137             self.keyb.angular.x = self.tilt
138             self.keyb.angular.y = self.cam_pan
139             self.keyb.angular.z = self.yaw_velocity
140             self.pubCommandP.publish(self.keyb)
141             rospy.loginfo(self.roll)
```

Programación del vehículo 5

*keyboard.py

```
141                 rospy.loginfo(self.roll)
142                 rospy.loginfo(self.pitch)
143                 rospy.loginfo(self.z_velocity)
144                 rospy.loginfo(self.yaw_velocity)
145
146             else:
147                 self.pubCommandP.publish(self.keyb)
148
149
150         def keyReleaseEvent(self,event):
151             key = event.key()
152
153             # If we have constructed the drone controller
154             #if not event.isAutoRepeat():
155             if not event.isAutoRepeat():
156                 # Note that we don't handle the release o
157                 # Now we handle moving, notice that this
158                 if key == KeyMapping.YawLeft:
159                     self.yaw_velocity -= 0.3
160
161                 elif key == KeyMapping.YawRight:
162                     self.yaw_velocity -= -0.3
163
164                 elif key == KeyMapping.PitchForward:
165                     self.pitch -= 0.3
166
167                 elif key == KeyMapping.PitchBackward:
168                     self.pitch -= -0.3
169
170                 elif key == KeyMapping.RollLeft:
171                     self.roll -= 0.3
172
173                 elif key == KeyMapping.RollRight:
174                     self.roll -= -0.3
175
176                 elif key == KeyMapping.IncreaseAltitude:
```

Programación del vehículo 6

*keyboard.py

```
176                 elif key == KeyMapping.IncreaseAltitude:
177                     self.z_velocity -= 0.3
178
179                 elif key == KeyMapping.DecreaseAltitude:
180                     self.z_velocity -= -0.3
181
182                 elif key == KeyMapping.Release:
183                     self.override = 0
184                     self.pubOverride.publish(self.override)
185
186                 self.keyb.linear.x  = self.pitch
187                 self.keyb.linear.y  = self.roll
188                 self.keyb.linear.z  = self.z_velocity
189                 self.keyb.angular.x = self.tilt
190                 self.keyb.angular.y = self.cam_pan
191                 self.keyb.angular.z = self.yaw_velocity
192                 self.pubCommandP.publish(self.keyb)
193
194 if __name__=='__main__':
195     import sys
196     rospy.init_node('keyboard')
197     app = QtGui.QApplication(sys.argv)
198     # Firstly we setup a ros node, so that we can commu
199     #rospy.init_node('bebopKeyController')
200     # Now we construct our Qt Application and associate
201     #t1 = SoloCamera()
202     #t1.start()
203     #controller = BasicBebopController()
204     #rate = rospy.Rate(10)
205     display = MainWindow()
206     display.show()
207     status = app.exec_()
208
209     # and only progresses to here once the application h
210     rospy.signal_shutdown('Great Flying!')
211     sys.exit(status)
```

Programación del vehículo 7

interpreter.py

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('bebop_driver')
import rospy
from geometry_msgs.msg import Twist
from std_msgs.msg import Empty
from std_msgs.msg import Int8
from bebop_msgs.msg import CommonCommonStateBatteryStateChanged

COMMAND_PERIOD = 20 #ms
class DroneController(object):
    def __init__(self):
        # Holds the current drone status
        self.status = -1
        # Allow the controller to publish to the /bebop/takeoff, land and reset topics
        self.pubLand    = rospy.Publisher('/bebop/land',Empty,queue_size=1000)
        self.pubTakeoff = rospy.Publisher('/bebop/takeoff',Empty,queue_size=1000)
        self.pubCommandPilot = rospy.Publisher('/bebop/cmd_vel',Twist,queue_size=1000)
        self.pubCommandCamera = rospy.Publisher('/bebop/camera_control',Twist,queue_size=1000)

        # Setup regular publishing of control packets
        self.command = Twist()
        self.command2 = Twist()

        # Land the drone if we are shutting down
        rospy.on_shutdown(self.SendLand)

    def SendTakeoff(self):
        # Send a takeoff message to the bebop driver Note we only send a takeoff message if the
not good!
        #if(self.status == DroneStatus.Landed):
        self.pubTakeoff.publish(Empty())

    def SendLand(self):
        # Send a landing message to the bebop driver Note we send this in all states, landing
        self.pubLand.publish(Empty())
```

Programación del vehículo 8

interpreter.py

```
35
36      def SendEmergency(self):
37          # Send an emergency (or reset) message to the bebop driver
38          self.pubReset.publish(Empty())
39
40      def SetCommandPilot(self,roll,pitch,yaw_velocity,z_velocity):
41          # Called by the main program to set the current command
42          self.command.linear.x  = pitch
43          self.command.linear.y  = roll
44          self.command.linear.z  = z_velocity
45          self.command.angular.z = yaw_velocity
46          self.pubCommandPilot.publish(self.command)
47
48      def SendCommand(self,event):
49          # The previously set command is then sent out periodically if the drone is flyi
50          #if self.status == DroneStatus.Flying or self.status == DroneStatus.GotoHover o
51          self.pubCommandPilot.publish(self.command)
52          self.pubCommandCamera.publish(self.command)
53
54      def SetCommandCamera(self, tilt=0, cam_pan=0):
55          self.command2.angular.y  = tilt
56          self.command2.angular.z  = cam_pan
57          self.pubCommandCamera.publish(self.command2)
58
59      def callback(self, cmd):
60          rospy.loginfo("Received a /cmd_vel message!")
61          rospy.loginfo(cmd)
62          print self.MsgBat
63          #rospy.loginfo("Linear Components: [%f, %f, %f]"%(cmd.linear.x, cmd.linear.y, c
64          #rospy.loginfo("Angular Components: [%f, %f, %f]"%(cmd.angular.x, cmd.angular.y
65          drone.SetCommandPilot(cmd.linear.y, cmd.linear.x, cmd.angular.z, cmd.linear.z )
66          drone.SetCommandCamera(cmd.angular.x, cmd.angular.y)
67          #drone.SendCommand()
68
69          #comm.SendCommand(env)
70          #pubLand    = rospy.Publisher('/bebop/land',Empty,queue_size=1000)
```

Programación del vehículo 8

interpreter.py

```
70          #pubLand    = rospy.Publisher('/bebop/land',Empty,queue_s
71          #pubTakeoff = rospy.Publisher('/bebop/takeoff',Empty,queu
72          #pubCommandPilot = rospy.Publisher('/bebop/cmd_vel',Twist
73          #command = Twist()
74          #self.command.linear.x  = msg.linear.x
75          #self.command.linear.y  = msg.linear.y
76          #self.command.linear.z  = msg.linear.z
77          #self.command.angular.z = msg.angular.z
78          #pubCommand.publish(command)
79
80      def takeoff(self, toff):
81          #rospy.Subscriber("/keyboard/takeoff", Int8, takeoff)
82          rospy.loginfo(toff.data)
83          #if toff.data== True:
84          if toff.data == 1:
85              drone.SendTakeoff()
86              rospy.loginfo("TAKE OFF")
87
88      def land(self, ld):
89          #rospy.Subscriber("/keyboard/land", Int8, land)
90          rospy.loginfo(ld.data)
91          #if ld.data== True:
92          if ld.data == 1:
93              drone.SendLand()
94              rospy.loginfo("LAND")
95              #print "LAND"
96
97      def battery_status(self, data):
98          #self.statusBar().showMessage(data.percent)
99          #rospy.loginfo("NIVEL DE BATERIA %s", str(data.percent))
100         self.MsgBat = "Bateria: "+ str(data.percent) + "%"
101         print self.MsgBat
102
103 def listener():
104     rospy.init_node('cmd_vel_listener')
105     rospy.Subscriber("/keyboard/cmd_vel", Twist, drone.callback)
```

Programación del vehículo 8

```
103 def listener():
104     rospy.init_node('cmd_vel_listener')
105     rospy.Subscriber("/keyboard/cmd_vel", Twist, drone.callback)
106     rospy.Subscriber("/keyboard/takeoff", Int8, drone.takeoff)
107     rospy.Subscriber("/keyboard/land", Int8, drone.land)
108     rospy.Subscriber('/bebop/states/common/CommonState/BatteryStateChanged', CommonCommonStateBatteryStateChanged,
    drone.battery_status)
109
110     #pubReset    = rospy.Publisher('/bebop/reset',Empty)
111     # Allow the controller to publish to the /cmd_vel topic and thus control the drone Setup regular publishing of
112     rospy.spin()
113
114 if __name__ == '__main__':
115     drone = DroneController()
116     listener()
```

Python ▾ Anchura de la pestaña: 4 ▾

4. ORB-SLAM2

El programa de ORB-SLAM2 esta orientada a la localización mediante la visión. Para ello es necesario tener el programa e instalarlo de la siguiente manera:

Primero se instalará las dependencias para lanzar los diferentes hilos del sistema SLAM.

sudo apt-get install libboost-all-dev
sudo apt-get install libblas-dev
sudo apt-get install liblapack-dev
sudo apt-get install libeigen3-dev

ORB- SLAM es una solución versátil y precisa para sistemas monoculares, capaz de calcular en tiempo real la trayectoria de la cámara y una reconstrucción 3D escasa de la escena en una amplia variedad de entornos , que van desde pequeñas secuencias de mano para un coche conducido alrededor de varios ciudad bloques . Es capaz de cerrar grandes bucles y realizar localización global en tiempo real y desde la línea de ancho.

Segundo pasamos a la clonación del código para el SLAM dentro de la carpeta /home/user/bebop_ws/src

git clone https://github.com/raulmur/ORB_SLAM2.git ORB_SLAM2

Instalación

3. Add the path where you cloned ORB-SLAM to the `ROS_PACKAGE_PATH` environment variable. To do this, modify your .bashrc and add at the bottom the following line (replace PATH_TO_PARENT_OF_ORB_SLAM):

```
export ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:PATH_TO_PARENT_OF_ORB_SLAM
```

4. Build g2o. Go into `Thirdparty/g2o/` and execute:

```
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
make
```

Tip: To achieve the best performance in your computer, set your favorite compilation flags in line 61 and 62 of `Thirdparty/g2o/CMakeLists.txt` (by default -03 -march=native)

5. Build DBoW2. Go into Thirdparty/DBoW2/ and execute:

```
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
make
```

Tip: Set your favorite compilation flags in line 4 and 5 of `Thirdparty/DBoW2/CMakeLists.txt` (by default -03 -march=native)

6. Build ORB_SLAM. In the ORB_SLAM root execute:

If you use ROS Indigo, remove the depency of opencv2 in the manifest.xml.

Instalación

6. Build ORB_SLAM. In the ORB_SLAM root execute:

If you use ROS Indigo, remove the depency of opencv2 in the manifest.xml.

```
mkdir build
cd build
cmake .. -DROS_BUILD_TYPE=Release
make
```

Tip: Set your favorite compilation flags in line 12 and 13 of `./CMakeLists.txt` (by default -03 -march=native)

4. Usage

See section 5 to run the Example Sequence.

1. Launch ORB-SLAM from the terminal (`roscore` should have been already executed):

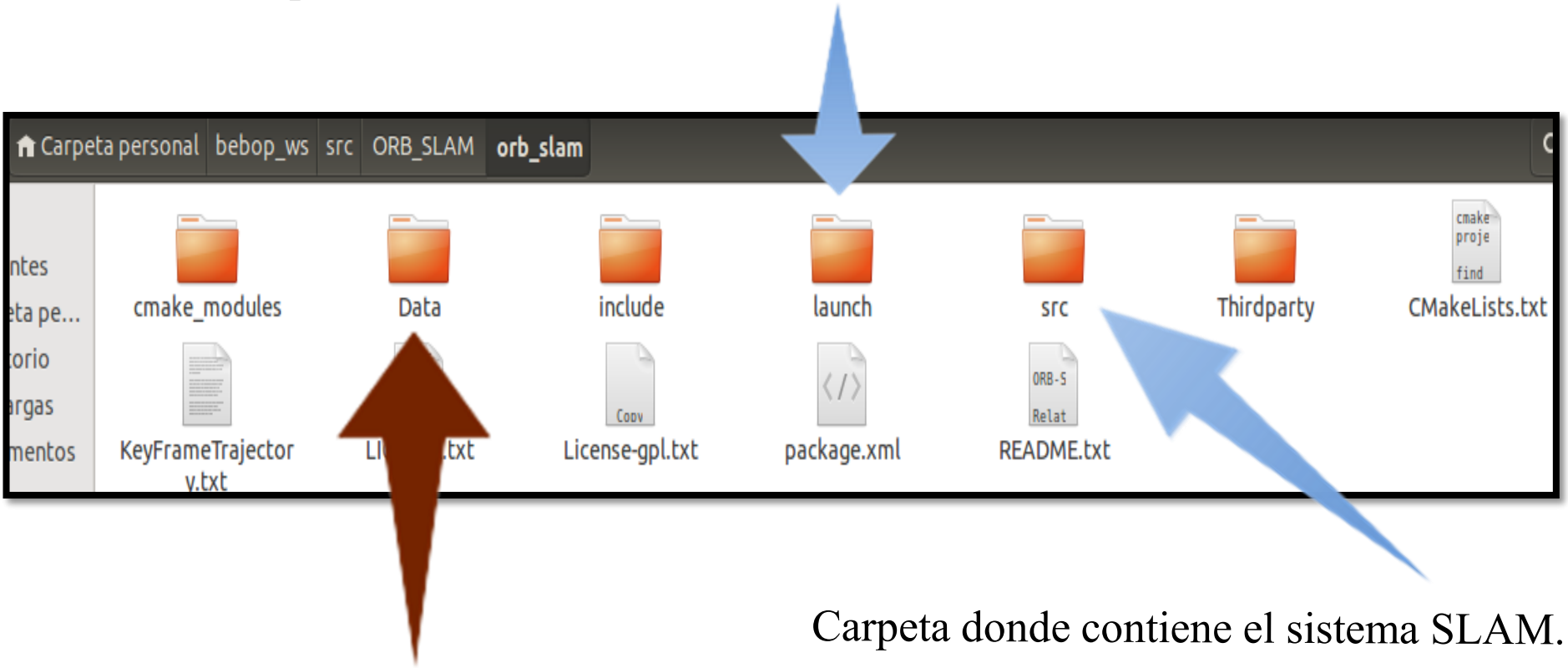
```
rosrun ORB_SLAM ORB_SLAM PATH_TO_VOCABULARY PATH_TO_SETTINGS_FILE
```

You have to provide the path to the ORB vocabulary and to the settings file. The paths must be absolute or relative to the ORB_SLAM directory.

We already provide the vocabulary file we use in `ORB_SLAM/Data/ORBvoc.txt.tar.gz`. Uncompress the file, as it will be

Instalación

Carpeta donde estarán los lanzadores.



Carpeta donde contiene el sistema SLAM.

Carpeta de suma importancia puesto que aquí van los parámetros de la calibración que se debe realizar a la que se este usando.

Lanzador

bebop2_load_orb2.launch

```
<?xml version="1.0"?>
<launch>
<!-- Launch the Bebop node-->
    <group ns="bebop">
        <node pkg="bebop_driver" name="bebop_driver" type="bebop_driver_node" output="screen">
            <param name="camera_info_url" value="package://bebop_driver/data/bebop_cameraCalib_opencv2.yaml" />
            <rosparam command="load" file="$(find bebop_driver)/config/defaults.yaml" />
        </node>
    </group>

<!-- Launch the bebop ORB-slam -->
    <node pkg="ORB_SLAM2" type="Mono" name="orb_slam2_mono"  args="/home/robertoiv/bebop_ws/src/ORB_SLAM2/Vocabulary/ORBvoc.txt /home/robertoiv/bebop_ws/src/ORB_SLAM2/Examples/Monocular/TUM1.yaml true" cwd="node" output="screen">
        <remap from="/camera/image_raw" to="/bebop/image_raw" />
    </node>

<!-- Launch the pose and Keyboard -->
    <!--node name="tf_pose" pkg="tf_pose" type="tf_pose" output="screen"/-->
    <!--node name="bebop_keyboard" pkg="bebop_driver" type="bebop_keyboard.py" required="true" output="screen"/-->

<node name="keyboard" pkg="bebop_driver" type="keyboard.py" required="true" output="screen"/>

<node name="interpreter" pkg="bebop_driver" type="interpreter.py" required="true" output="screen"/>

</launch>
```

Parámetros de la cámara

bebop_cameraCalib_opencv2.yaml

```
%YAML:1.0

# Camera Parameters. Adjust them!

# Camera calibration parameters (OpenCV)
#

Camera.fx: 409.9343948310273
Camera.fy: 386.86372211977493
Camera.cx: 316.60806660824585
Camera.cy: 175.50740246286716

# Camera distortion paremeters (OpenCV) --
Camera.k1: 0.049795
Camera.k2: -0.093157
Camera.p1: -0.012412
Camera.p2: 0.000269

# Camera frames per second
Camera.fps: 50.0

# IR projector baseline times fx (aprox.)
Camera.bf: 40.0

# Color order of the images (0: BGR, 1: RGB. It is ignored if images are grayscale)
Camera.RGB: 1

# Close/Far threshold. Baseline times.
ThDepth: 50.0

# Deptmap values factor
DepthMapFactor: 1000.0

#---------------------------------------------------------------------------
# ORB Parameters
#---------------------------------------------------------------------------
```

Parámetros de la cámara

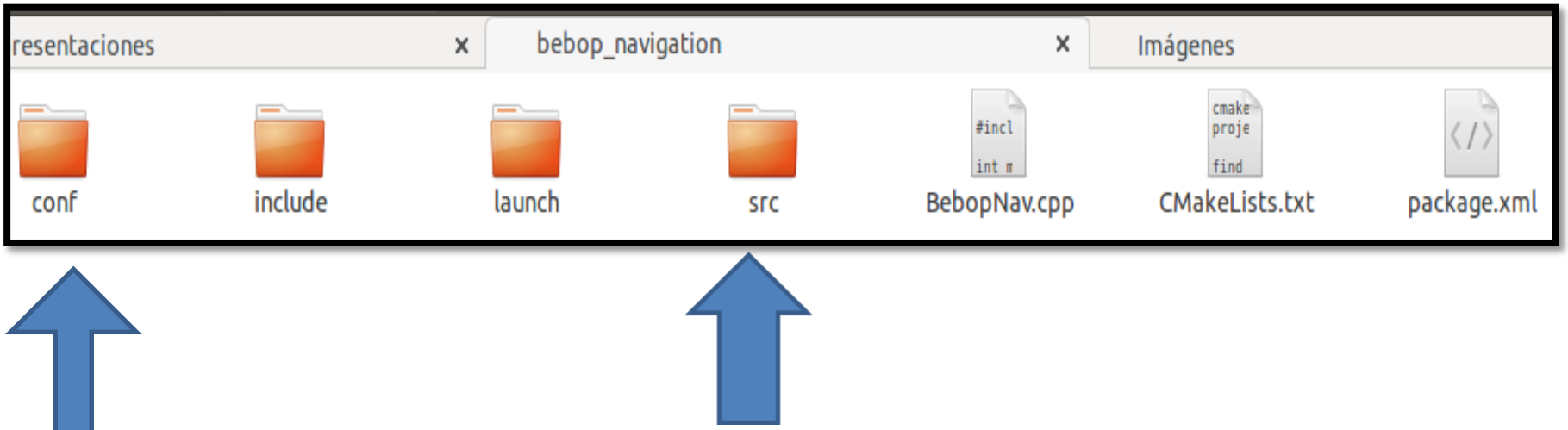
bebop_cameraCalib_opencv2.yaml

```
# Deptmap values factor
DepthMapFactor: 1000.0

#--------------------------------------------------------------------------------------------
# ORB Parameters
#--------------------------------------------------------------------------------------------

# ORB Extractor: Number of features per image
ORBextractor.nFeatures: 1000

# ORB Extractor: Scale factor between levels in the scale pyramid
ORBextractor.scaleFactor: 1.2

# ORB Extractor: Number of levels in the scale pyramid
ORBextractor.nLevels: 8

# ORB Extractor: Fast threshold
# Image is divided in a grid. At each cell FAST are extracted imposing a minimum response.
# Firstly we impose iniThFAST. If no corners are detected we impose a lower value minThFAST
# You can lower these values if your images have low contrast
ORBextractor.iniThFAST: 20
ORBextractor.minThFAST: 7

#--------------------------------------------------------------------------------------------
# Viewer Parameters
#--------------------------------------------------------------------------------------------
Viewer.KeyFrameSize: 0.05
Viewer.KeyFrameLineWidth: 1
Viewer.GraphLineWidth: 0.9
Viewer.PointSize:2
Viewer.CameraSize: 0.08
Viewer.CameraLineWidth: 3
Viewer.ViewpointX: 0
Viewer.ViewpointY: -0.7
Viewer.ViewpointZ: -1.8
Viewer.ViewpointF: 500
```

5. PID

El control PID (Proporcional, Integral, Derivativo) es un sistema que suministra entradas de control que son proporcionales a la diferencia entre las salidas actuales del sistema del vehículo aéreo y los valores de referencia del sistema SLAM. En la siguiente ecuación se muestra de manera generalizada los controladores PID en su forma matemática (1). Introduciendo una retroalimentación se compensará perturbaciones como: desequilibrio y desorientaciones.

Desarrollo del PID



Carpeta donde se realizará el control PID.

Carpeta contenedora del nodo para la publicación de la posición del vehículo.

Vuelo Semi-Autónomo

Ahora que ya saben como mandar comandos al drone, leer su posición mediante del sistema Slam, y agregarle un controlador por medio de la retroalimentación de lo que la cámara observa. Pasamos a ensamblar los código para poder lanzar nuestro sistema Semi-Autónomo por medio de WayPoints.