Ahmet's Robots

Nicola Roberto Zema

<2017-09-06 mer>

Contents

1 `ebug_swarm_control`

1.1 Instructions

Just drop the `ebug_swarm_control` folder into `/home/<username>/catkin_ws/src` and issue a `catkin_make` as per manuals/tutorials.

I have made also a launch file.

Jeez!

1.2 Mail

Original mail from the man.

```
   Hi Nicola,

   Can we build a ROS structure consisting of the following blocks:

1. control_node
2. localization_node
3. planning_node
4. visualization_node

They will basically do nothing but exchange information through the
```

ROS provided (blue marked in the attached diagram) mechanisms (I forgot the name of this whiteboard?).

If you can write a simple program in the planning_node that will generate an initial position x,y 0 < x < 640 and 0 < y < 480 (blob camera resolution).

control_node will read it and write it for localization_node to read.

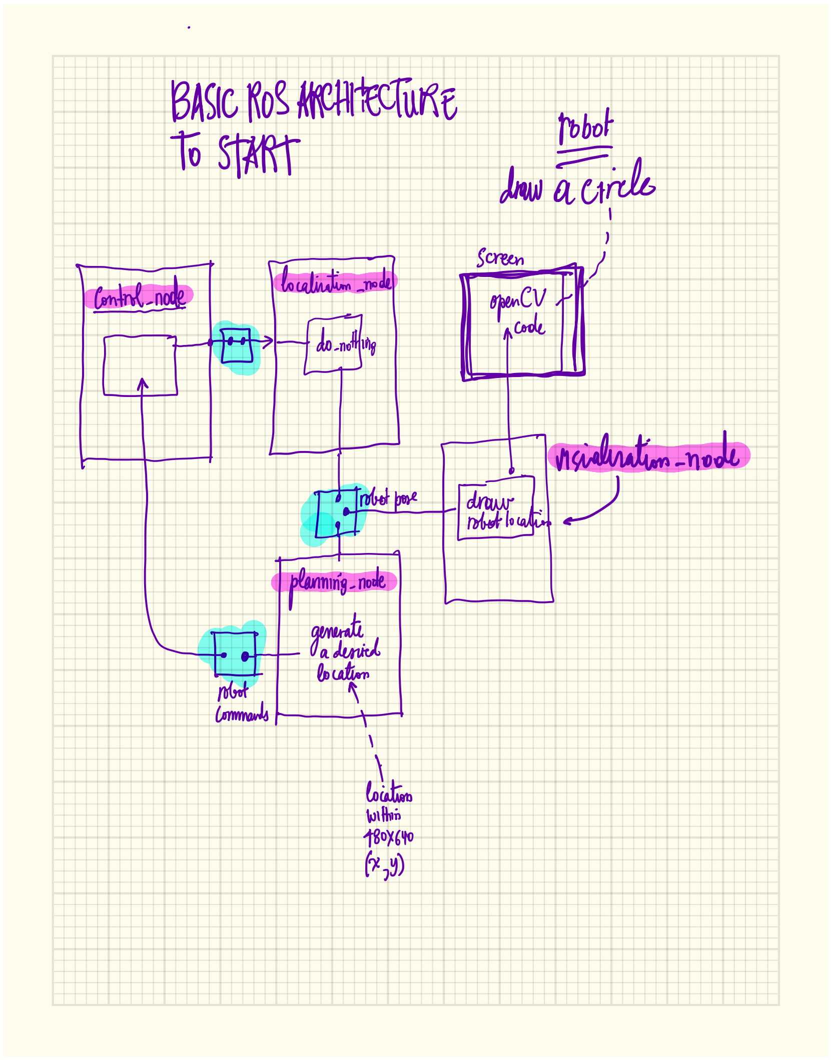
Then the localization_node will read and write it for visualization_node and planning_node to read.

visualization_node will have an openCV code to display a circle in a rectangular window representing the current position of the "robot" as a simple circle.

planning_node will read the position and generate a next position to model a random walk of the robot. This cycle will keep going indefinitely.

I think this will be a good starting point for us.

1.3 Attached Material



1.4 Action Plan

1. **DONE** Create nodes
2. **DONE** Setup makefiles and xml
3. **DONE** `planning_node`

```
the planning_node that will
generate an initial position x,y 0 < x < 640 and 0 < y < 480 (blob
camera resolution).
```

(a) **DONE** basic data generation

- It is an image `640x480`;
- It is a `sensor_msgs/Image Message`
- File: `sensor_msgs/Image.msg`
- Things to take into consideration:
- Header header (see 3(a)ii)
  - WTF?!?!
- `uint32 height         # image height, that is, number of rows`
  - Can Fake it
- `uint32 width          # image width, that is, number of columns`
  - Can Fake it
- `string encoding       # Encoding of pixels -- channel meaning, ordering, size`
  - Can Fake it
- `uint8 is_bigendian    # is this data bigendian?`
  - Can Fake
- `uint32 step           # Full row length in bytes`
  - Can Fake
- `uint8[] data          # actual matrix data, size is (step * rows)`
  - Can Fake:
    - Fill it with 1s
- **ROS TAKES CARE OF ALL DEFAULTS**

i. How to handle it Send it as a message on a dedicated topic.

ii. Info

```
# This message contains an uncompressed image
# (0, 0) is at top-left corner of image
#

Header header        # Header timestamp should be acquisition time of image
                     # Header frame_id should be optical frame of camera
                     # origin of frame should be optical center of cameara
                     # +x should point to the right in the image
                     # +y should point down in the image
                     # +z should point into to plane of the image
                     # If the frame_id here and the frame_id of the CameraInfo
                     # message associated with the image conflict
                     # the behavior is undefined

uint32 height         # image height, that is, number of rows
uint32 width          # image width, that is, number of columns

# The legal values for encoding are in file src/image_encodings.cpp
# If you want to standardize a new string format, join
# ros-users@lists.sourceforge.net and send an email proposing a new encoding.

string encoding       # Encoding of pixels -- channel meaning, ordering, size
                      # taken from the list of strings in include/sensor_msgs/i

uint8 is_bigendian    # is this data bigendian?
uint32 step           # Full row length in bytes
uint8[] data          # actual matrix data, size is (step * rows)
```

4. **DONE** `control_node`

   ```
   control_node will read it and write it for localization_node to read.
   ```

   (a) **DONE** Message Transport
       - Read the image from 3
       - Rebroadcast it.

5. **DONE** `localization_node`

   ```
   the localization_node will read and write it for
   visualization_node and planning_node to read.
   ```

   (a) **DONE** Message Transport Same as 4

6. **DONE** `visualization_node`

   ```
   visualization_node will have an openCV code to display a circle in a
   rectangular window representing the current position of the "robot" as
   a simple circle.
   ```

   (a) **DONE** Data Endpoint Just print out some stuff.

2 OLD

2.1 Original Request

1. Important stuff

   (a) Github repo
       `https://github.com/monash-wsrn/ebug2014-qt`

2. Other mail

   Erwin has disentangled the localization routines from the of others (He was a great student, works at the Seagate R&D center now).

3. We have an independent program that processes the blob trace and reports the eBug positions and orientation. We have tested it with only one stationary eBug though. I will need to collect more trace data.

4. Here is an extract from his program: The eBug is nicely sitting at the position 665, 502 with an orientation angle 262 degrees :-)

5. I will need to check out the camera view 0,0 point and the coordinates of other corners to determine the limits. Also will work out the reported angle's reference angular position.

6. ID = 4 x = 664.927 y = 502.462 angle = 262.533 ID = 4 x = 664.991 y = 502.46 angle = 262.393 ID = 4 x = 664.722 y = 502.149 angle = 262.707 ID = 4 x = 664.823 y = 502.443 angle = 262.484 ID = 4 x = 664.898 y = 502.411 angle = 262.479 ID = 4 x = 664.868 y = 502.622 angle = 262.447 ID = 4 x = 664.765 y = 502.315 angle = 262.713 ID = 4 x = 664.825 y = 502.456 angle = 262.642 ID = 4 x = 665.045 y = 502.419 angle = 262.406 ID = 4 x = 664.864 y = 502.598 angle = 262.393 ID = 4 x = 664.943 y = 502.307 angle = 262.572

7. So ... I think eBug localization with RGB LED tracking is under control. If you can, I will need your continuing help on trajectory control block.

2.2 New Robots

1. Mail 1 Hi Nicola,

   I attach my thinking about the ROS based architecture of the eBug Swarm Control System (running on Linux). There are three independent processes (in ROS parlance, "nodes", I think).

2. Big rectangle at the left is the only one that has connection with the radio, and gets camera blobs, sends eBug control commands. This one is pretty much under control.

3. Small rectangle at the upper right corner is the one that we attempted to extract from Erwin's code. I'm trying to contact Erwin to see whether I can get him to do it.

4. Small rectangle at the lower right is the place perhaps you can help. What I want at the moment is a small demo, say 5 eBugs that to do something interesting. Simplest possibility I can think of is to pre-calculate trajectories for each eBug in a way that they won't collide and send periodical commands to maintain the desired trajectories. Do you think you can help with this?

5. A more challenging approach would be to extract control modules from Erwin's code and reuse withing the ROS context I have drawn.

6. Please let me know what you think.

7. Material `material/scan_asekerci_2017-09-04-16-07-11.pdf`
8. Python files
   (a) nrf.py library and a short sample python code
   `./nrf.py`
   `./discover.py`