Turtlebot Navigation

Xiyang Dai

05/17/2012

Introduction

This project is intended to provide basic library to do Turtlebot navigation. Turtlebot is a developing kit combining a Roomba robot, a Kinect camera and a robot arm. It is especially great for doing indoor robot development.

In this reference, a simple turtlebot navigation process is described. Main work consists Kinect calibration, data fusion odometry and navigation. The whole codes can be found in nyu-robotics github at `https://github.com/ylecun/nyu-robotics` Great thanks to Professor Yann LeCun for providing huge help during the project.

1 Kinect Calibration

The openni Kinect driver provides default camera models with reasonably accurate focal lengths which relate 3D points to 2D image coordinates. However it does not model lens distortion and rgb depth calibration. In order to get corresponding RGBD data, we need do rgb depth calibration ourselves.

The basic idea to calibrate rgb and depth camera is the same as stereo camera calibration. However, one thing should be noticed is that the Kinect depth camera is actually an IR camera which initially programmed to show only depth information. We cannot use depth image to do pattern recognization, so we need to hack the Kinect to use IR image instead (see figure 1). Then we can do calibration based on RGB and IR images of chessboard patterns with related OpenCV's calibration routines [1]. A more detailed tutorial of how to calibrate Kinect can be found in Burrus's project [2]. Also, an automated Kinect calibration program is under development.

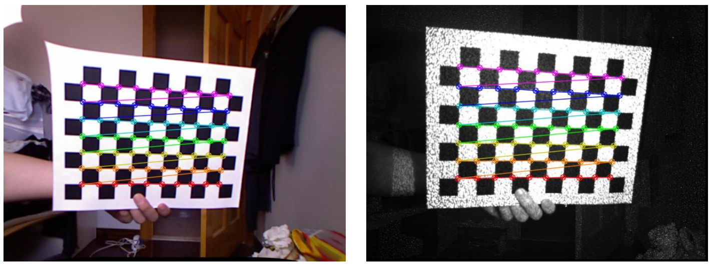
(a) RGB image (b) IR image

Figure 1: The Kinect calibration using RGB and IR image

2 Data Fusion Odometry

2.1 Roomba Hack

Turtlebot uses an iRobot Creat as a mobile base. iRobot Creat is a simplified edition of roomba vacuum robot which provides multiple sensors and uses serial port to communicate with computer. In our project, we need do some roomba hacks to get wheel sensor data and gyro data. These functions have been integrated into room-odometry library. Two major functions are

```
;;Create new odometry object
(setq od (new odometry 0 0 0 0))
;;Update gyro sensor data
(==> od gyro-update rb)
;;Update wheel sensor data
(==> od wheel-update rb)
```

Details about how to hack roomba can be found in book "Hacking Roomba" [3].

2.2 Visual Odometry

The SURF feature is used to implement visual odometry. First presented in ECCV 2006 by Herbert Bay [4], SURF feature is claimed much faster than SIFT and good for computer vision tasks like object recognition and 3D reconstruction. In our implementation of visual odometry, SURF feature is applied to select and track interested points in RGB space (see figure 2). Then, matched keypoints are projected to RGBXYZ space. Finally, basic image registration idea is used to calculate rotation matrix and transform matrix through a linear least square solver.

We implement SURF match algorthm based on OpenCV implementations. In order to wrap newest OpenCV libarary, a new OpenCV configuration file is provided. Related files can be found in ”local/opencv2/”.

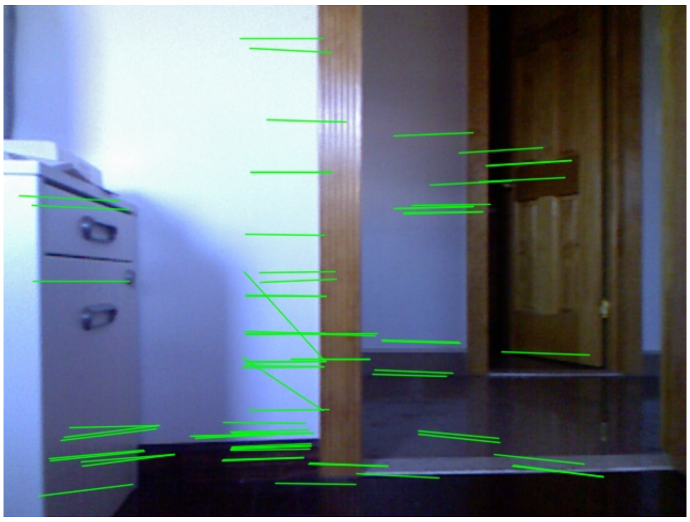

Figure 2: Keypoints match using SURF feature in RGB space

The whole visual odometry program has been integrated into roomb-odometry library and the major function is:

```
;;Update visual odometry
(==> od vo-surf-update current-image last-image rgbxyz-data)
```

2.3 Data Fusion

A two-level kalman filter is involved to combine the wheel sensor data, the gyro sensor data and the visual odometry data. The total process of data fusion is described in figure 3. The kalman filter is implemented based on OpenCV implementation which can be found in ”local/opencv2/”.

The complete process of calculating odometry is:

```
...
(while 1
    ...
    ;;Get robot position
    (==> od get-position-x)
    (==> od get-position-y)
    (==> od get-position-angle)
```

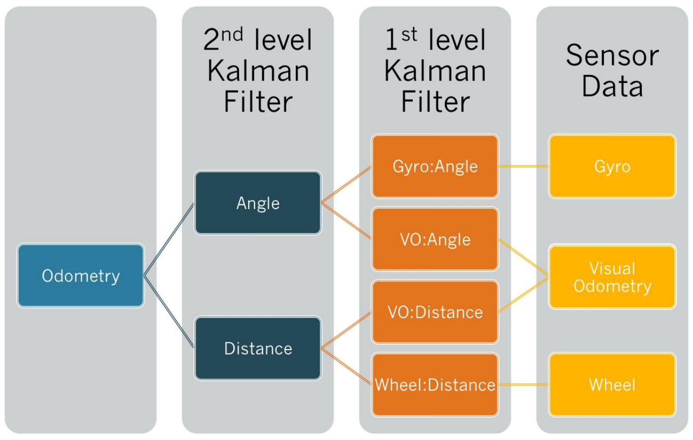

Figure 3: The process of two level's kalman filter data fusion

```
;;Display odometry image at position(0,0)
(==> od display 0 0)
...
;;Update gyro sensor data
(setq data1 (==> od gyro-update rb))
;;Update wheel sensor data
(setq data2 (==> od wheel-update rb))
;;Update visual odometry
(setq data3 (==> od vo-surf-update current-image last-image rgbxyz-data))
;;Fusion data
(==> od update data1 data2 data3)
...
())
```

3 Navigation

In the navigation part, a cost map based path planning is implemented. The process is described as following:

1. The ground truth is calculated by RGBXYZ data using least square plain fitting.

2. A cost map is calculated by RGBXYZ data based on the ground truth.

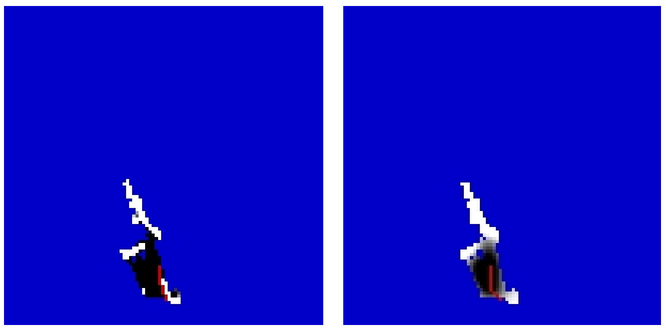
(a) Cost map before adding safe margin (b) Cost map after adding safe margin

Figure 4: Adding safe margin to cost map

3. A modified A* algorithm is involved to do path planning on the cost map

Notice that using A* path planning may sometimes lead robot too close to obstacle. Hence a simple safe margin trick is introduced to improve local path planning safety (see figure 4). Inspired by lazy image blur algorithm, our simple adding-safe-margin algorithm is described as following.

Algorithm 1 Add safe margin

```
$Source \leftarrow CostMap$
$Scale \leftarrow MarginScale$
while $Scale > 0$ do
  for $i = 1$ to Width($CostMap$) do
    for $j = 1$ to Height($CostMap$) do
      $CostMap(i,j) \leftarrow CostMap(i,j) + Source(i+1,j+1) + Source(i,j+1) + Source(i-1,j+1) + Source(i+1,j) + Source(i,j) + Source(i-1,j) + Source(i+1,j-1) + Source(i,j-1) + Source(i-1,j-1)$
    end for
  end for
  Normalize($CostMap$)
  $Scale \leftarrow Scale - 1$
end while
```

Finally, a simple driving function is implemented based on the path planning result. The major functions of navigation process are:

```
...
;;Calculate cost map
(map2cost mp costmap 0.2)
;;Add safe margin to costmap
(add-safe-margin costmap)
;;Do A* path planning
(setq direction (costmap2planning costmap 0.2 50 50))
;;Drive turtlebot
(drive r direction)
...
```

References

[1] http://opencv.willowgarage.com/documentation/cpp/camera_calibration_and_3d_reconstruction.html

[2] http://nicolas.burrus.name/index.php/Research/KinectCalibration

[3] Kurt, T.E., *Hacking Roomba: ExtremeTech*, Wiley Pub, 2006.

[4] H. Bay, T. Tuytelaars, L. Van Gool, *SURF: speeded up robust features*, in: ECCV, 2006.