Sensor Applications in Dynamic Situations

By: Ezra Idy

Mechatronics, Controls, & Robotics Lab
Overview

• Evolution of Project
• Project #1: Sensor Fusion of Angular Rotation Test Bed
• Project #2: Onboard Vision System for Swarm Robots
Evolution of Project

**Past:**
- Planned on using Boe-Bots
- Fusion between Camera and Optical Encoder
- Centralized System

**Present:**
- Working with TurtleBot3
- Fusion between Camera and LIDAR
- Decentralized System
Background

What is Sensor Fusion?
The combination of data from several sensors for the purpose of improving application or system performance.

3 distinct types:
• Competitive Fusion – independent measurements of the same property
• Complementary Fusion – more complete view of object
• Cooperative Fusion – derived information to obtain completely new information
Project #1: Sensor Fusion of Angular Rotation Test Bed
Objective

• Create an easy, affordable, and accessible Sensor Fusion system
• Analyze the data obtained from:
  • Individual sensors
  • Multiple sensors
Sensor Fusion test bed

- Battery Pack
- Parallax Continuous Servo Motor
- Raspberry Pi & Shield
- Raspberry Pi Camera
- AprilTag
- 8-bit encoder
RQT Graph
Camera

• The Camera is used to detect the tags orientation
  • Converted Quaternion into Roll, Pitch, Yaw
• The rate of the Camera is 60 Hz
• High Resolution images
• Slow but accurate
• Lighting issues when detecting

```cpp
void messageReceived(const apriltags_ros::AprilTagDetectionArray::ConstPtr& msg) {
    int tags_detected;
    int tag_id;
    tags_detected = msg->detections.size();
    if (tags_detected == 0){
        //ROS_INFO("NO TAGS FOUND");
    }

tag_detection_array.detections = msg->detections;

    //sending detections
    for (int i = 0; i < tag_detection_array.detections.size(); i++){

        tag_id = tag_detection_array.detections[i].id;
        if (tag_id == 11){
            testTagPose = tag_detection_array.detections[i].pose;

            //quaternion form
            x = testTagPose.pose.orientation.x;
            y = testTagPose.pose.orientation.y;
            z = testTagPose.pose.orientation.z;
            w = testTagPose.pose.orientation.w;
        }
        tf::Quaternion q(x, y, z, w);
        tf::Matrix3x3 m(q);
        m.getRPY(roll, pitch, yaw);
        if (yaw < 0) {
            angle = yaw*(180/pi) + 360;
        } else{
            angle=yaw*(180/pi);
        }
        if (record==0){
            angle_0=angle;
            record = 1;
        } else{ angle=angle_0;
        }
        if (angle==angle_0){
            rel_angle = angle-angle_0;
        } else{
            rel_angle = angle-angle_0+360;
        }
    }
}
```
Encoder

- 16 ticks = 1 rotation or 1 tick = 22.5 degrees
- Ticks only increase
  - Need to increase and decrease angle
- Rate at 120 Hz
- Fast but can accumulate error over time
  - Issue when quickly changing directions

```c
while(ros::ok())
{
    std_msgs::Int16 test;
    //new_position = left_rotation=0*pi;
    //position[1]=new_position;
    ang[1]=angle;
    Cam[1]=Cam_Ang;
    if (speed == 101,100,98,97,96){
        speed = base;
    }
    if(speed>base){
        //ROS_INFO("position 0: %f",position[0]);
        //ROS_INFO("position 1: %f",position[1]);
        //distance[1]=position[1]-position[0];
        distance[1]=ang[1]-ang[0];
        if (distance[1]>0){
            total_angle += distance[1]=(Cam_Ang-total_angle);
        }
        if (total_angle >= 360){
            total_angle = 0;
        }
    }
    if(speed<base){
        distance[1]=ang[0]-ang[1];
        //distance[1]=position[0]-position[1];
        if (distance[1]<0){
            total_angle += distance[1]=(Cam_Ang-total_angle);
        }
        if (total_angle <= 0){
            total_angle = 360;
        }
    }
    if (speed==base || (speed <182 && speed >99) && abs(Cam[0]-Cam[1])>2*rad){
        //if (abs(Cam_Ang-total_angle)>error){
            total_angle=Cam_Ang;
            ROS_INFO("corrected angle: %f",total_angle);
        //}
    }
    ROS_INFO("angle total: %f",total_angle);
    Cam[0]=Cam[1];
    ang[0]=ang[1];
    //position[0]=position[1];
    distance[0]=distance[1];
```
Entrepreneurship

• Project was chosen for the Stern Entrepreneurship collaboration
• The Stern students are currently validating customers
• Testing the educational market place
  • In contact with CIJE (Center for Initiatives in Jewish Education)
  • In contact with ITEST
Application

- Educational purpose – Can be used to teach sensor fusion on a basic level
- Interactive learning – Learn Kalman filtering and PID control
- Embedded vs remote – Learn about the different sensor types and how they effect a system
Future Work

• Fine tune the code
  • Implement Kalman Filtering to the system
  • Implement PD/PID controls

• Plan on further developing GUI using smartphone device
  • Through ROS and Xcode
Kalman Filtering

• Applying competitive fusion
• Implement Kalman filtering that will add weighted values to the different data obtained
• Use PID tuning on testbed for user control
• Incorporated in the app
Graphic User Interface

• Currently have interface template
• Plan on using ROS and Xcode to further design
• Allow for the selection of options:
  • Fusion
  • Encoder
  • Vision
• Can choose desired angles
• Easy testing
Project #2:
Onboard Vision System for Swarm Robotics
Objective

• Incorporate vision on swarm system
• Compare and evaluate different camera methods
  • 360 camera vs Servo camera
• Identify objects obtained from LIDAR reading
  • Robot vs Obstacle
Background - Swarm System

• Centralized vs Decentralized system
• System that can be supervised on an abstract level
• Robust system
• Maintaining connected formations
• Robots can facilitate mapping and localization
• Path planning and obstacle avoidance
Problems Faced

• Not enough storage → increased Swapfile size
• A lot of processing power required → still images
• Latency → rospicam_node
360 Camera

Pro(s):
• Detects tags faster
• Wire is steady

Con(s):
• Range
• Permanent
• Poor Resolution
RQT Graph
Camera Node

- Still Image
- Not using Raspicam_node
De-warping

```python
import rospy
from std_msgs.msg import String
from sensor_msgs.msg import CompressedImage, CameraInfo, Image

import numpy as np
import time
import cv2
from cv_bridge import CvBridge

xmap = {}
ymap = {}
pub = None

def Map (Wx,Hx,Wd,Hd,R1,R2,Cx,Cy):
    map_x = np.zeros((Hd,Wd),np.float32)
    map_y = np.zeros((Hd,Wd),np.float32)
    for y in range(0,int(Hd-1)):
        for x in range(0,int(Wd-1)):
            r=(float(y)/float(Wd))*R2+R1
            theta = (float(x)/float(Hd))*np.pi*2+np.pi #shift warping so that it starts from 180, picture splits at back.
            xs = Cx+r*np.sin(theta)
            ys = Cy+r*np.cos(theta)
            map_x.letterset([y,x] % (Hd - 1),int(xs))
            map_y.letterset([y,x] % (Wd - 1),int(ys))
    return map_x, map_y

def de warp (img,xmap,ymap):
    return cv2.remap(img, xmap, ymap,cv2.INTER_LINEAR)
```
Apriltag detection

• Focus on pixel location in remapped image
  • Complex to use x, y, z coordinates system
• From pixel location, the angle of the identified object can be determined
• Obtains information for each tag detected
• Can find tag depth from camera
  • Find diagonal of tag from corner info
Servo Camera

Pros:
• Adaptable
• High Resolution
• Large Detection Range
  • ~6 feet

Cons:
• Wire can snap
• Wait time
• Calibrating additional Servo
Camera Node

• Using raspicam_node

• Originally used still image based off of raspicam_node
  • Grab image from raspicam_node when pinged
Servo Calibration

• Add motor identification
• Enable torque of motor
• Set max velocity
• Set RPM of motor
• Enable position control
• Set Baud rate
LIDAR Ping

• Sends Ping when unknown object is detected
  • New object enters system
  • Blobs merge then separate
  • Manual reset from user

• Outputs:
  • x, y, z position of object, object number, theta, identification

• Identification:
  • 0-7: Apriltag ID
  • 99: Unknown
  • 999: Confirmed obstacle
Apriltag Detection

- From Ping, servo moves to the angles of unknown objects
- After 2-3 seconds camera determines whether robot (detects tag) or obstacle (does not detect tag)
- Tags relay x, y, z position of robot and the tag ID
- The data obtained is then crosschecked with the LIDAR data
  - Needed to constrain the camera range
Future Work

• Set up 360 Camera on Turtlebot 3
• Compare the efficiencies of both systems
  • Servo Camera and 360 Camera