The University of Washington, Department of EE Technical Report Series

Jeff Bilmes , Travis Saling, Sekar Thiagarajan
{bilmes,trav,sekar}@ee.washington.edu

Dept of EE, University of Washington
Seattle WA, 98195-2500

UWEE Technical Report
Number UWEETR-2001-0000
January 2001

Department of Electrical Engineering
University of Washington
Box 352500
Seattle, Washington 98195-2500
PHN: (206) 543-2150
FAX: (206) 543-3842
URL: http://www.ee.washington.edu

The University of Washington, Department of EE Technical Report Series

Jeff Bilmes , Travis Saling, Sekar Thiagarajan
{bilmes,trav,sekar}@ee.washington.edu

Dept of EE, University of Washington
Seattle WA, 98195-2500

University of Washington, Dept. of EE, UWEETR-2001-0000
January 2001

Abstract

All important research labs maintain some kind of internal system for research documentation, typically in the form of a technical report series. The newly established UWEETR series will serve that purpose for our department. This short document explains some of the rational behind the series, provides steps on how to automatically submit articles to the series via the WWW, and offers suggestions about what type of documents should be submitted to the series.

1 Introduction to Elcano Project

The Elcano Project is developing low-cost hardware and software kits to convert any vehicle to self-drive. We are concentrating on recumbent tricycles, since that produces a real people mover at a total cost of under $5000. Our kits could also be used in full-sized cars or toy RC cars.

- $C2-$ **Dual control:** low level vehicle control, either from the driver or the AI.
- $C3-$ **Pilot:** Detects obstacles and feeds settings for the next path segment to C2.
- $C4-$ **Path Planner:** Computes the best route from current location to destination.
- $C5-$ Obstacle detection from sonars.
- $C6-$ **Navigation:** Navigator: Reads GPS, INU, Odometer, Compass etc. to get best position estimate.
- $C7-$ **Vision:** Locates certain features of interest (Raspberry Pi).

2 Navigation and Localization

2.1 Overview

Localization is a module that we use for the navigation of the trike. Trike should be able to send its location without using GPS. GPS only cannot be relied upon as satellite reception is poor indoors, tunnels and sometimes near tall buildings. The Dead Reckoning principle is used which calculates the position based on the previous position estimates. Dead Reckoning uses Odometer data from $C2\_Low\_Level$ and compass bearing data to calculate the position with respect to last position state of the trike. The trike will use other sensor data like INU and optical mouse to get the best estimated position in future. Localization system is connecting with $C2$ Dual Mode Arduino Mega and $C4$ Path Planner Arduino Nano. Fuzzy numbers algorithm is used to get the best estimated position of the trike

2.2 Dead Reckoning

Dead reckoning is implemented in the three-wheeled bicycle in order to overcome the limitations of GPS technology alone. Satellite microwave signals are unavailable in parking garages and tunnels, and often severely degraded in urban canyons and near trees due to blocked lines of sight to the satellites or multipath propagation. In a dead-reckoning navigation system, the trike is equipped with sensors that know the wheel circumference and record wheel rotations and steering direction. This can be read by the navigation system from the controller-area network bus. The navigation system then uses a fuzzy algorithm to integrate the always-available sensor data with the accurate but occasionally unavailable position information from the satellite data into a combined position fix.

2.3 Definition of Fuzzy Algorithm

2.3.1 Horizontal Dilution of Precision (HDOP)

The horizontal dilution of precision (HDOP) allows one to more precisely estimate the accuracy of GPS horizontal (latitude/longitude) position fixes by adjusting the error estimates according to the geometry of the satellites used. Theoretically, given the HDOP, one can obtain error estimates that are good for all fixes with that HDOP, rather than the more general error estimates for all position fixes (regardless of HDOP). In probability terminology, HDOP is an additional variable that allows one to replace the overall accuracy estimates with conditional accuracy ones for the given HDOP value.

2.4 Example of position sensor fusion

The previous estimate of trike bearing and position was sent as SENSOR $Pos3.50, 5.20Br22$. That is, the trike is at 3.5 m east, 5.2 m north and heading NNE. The trike travels for 0.5 s at a constant speed of 2.5 m/s maintaining the NNE heading. It has travelled 1.25 meter, and the new position from dead reckoning is $(3.5 + 1.25 * cos(68), 5.2 + 1.25 * sin(68)) = (3.968, 6.359)$. At this time a GPS reading returns a position of (4.75, 8.21). The GPS Horizontal Dilution of Precision (HDOP) is 0.75.
The horizontal dilution of precision (HDOP) allows one to more precisely estimate the accuracy of GPS horizontal (latitude/longitude) position fixes by adjusting the error estimates according to the geometry of the satellites used. Theoretically, given the HDOP, one can obtain error estimates that are good for all fixes with that HDOP, rather than the more general error estimates for all position fixes (regardless of HDOP). In probability terminology, HDOP is an additional variable that allows one to replace the overall accuracy estimates with conditional accuracy ones for the given HDOP value. **What should be our new estimate of position?**

We use the model where,

error = $\sqrt{((3.04 * HDOP)^2 + (3.57)^2)}$

We compute the GPS error as $\sqrt{((3.04 * 0.75)^2 + (3.57)^2)} = 4.236m$.

On the dead reckoning, assume bearing +/- 2 degrees, and speedometer is good to 4%. Distance traveled could be +/- 0.05m. The bearing error could be +/- 1.25*sin(2) = 0.044m. We need to translate to a coordinate system in the direction of travel. Set the starting point to the DR position. The dead reckoning estimate in this coordinate system is (0, 0). The GPS point in this coordinate system translates to (4.75-3.968, 8.21-6.359) = (0.782, 1.851). Rotate by the bearing: (0.782*cos(68)+1.851*sin(68), -0.782*sin(68)+1.851*cos(68)) = (2.009, -0.032). In direction of travel, the fuzzy numbers are 1.25 +/-0.05 and 2.009 +/- 4.236. The maximum fuzzy value for both is at approximately 1.26. In the perpendicular direction the dead reckoning number is 0 +/- 0.044. GPS is -0.032 +/- 4.236. Approximate intersection is at approximately -0.002.Thus the position estimate in the direction of travel is (1.26, -0.002). This point needs to be put back into the original coordinate system. Rotate it in the opposite direction: (1.26*cos(68)+0.002*sin(68), 1.26*sin(68)-0.002*cos(68)) = (0.47, 1.167). Adding back the origin gives (0.47+3.5, 1.167+5.2) = (3.97, 6.367).
New position is SENSOR Pos 3.97,6.367 Br 21

2.5 Assumptions

- Initial position when the GPS doesnt work: We assume the initial position $latitude = 47.62130, longitude = -122.35090$, speed is $0mmps$. When the GPS doesnt work, the system can use this initial position to calculate the location.

- Hard Coded values for GPS, Compass, and Distance errors:
  - $GPS\_ERROR((long)4236)$
  - $COMPASS\_ERROR : ((double)2)$
  - $DISTANCE\_ERROR : ((double)0.04)$

2.6 Required Libraries

- Adafruit_LSM303_U
- Adafruit_GPS-master
- Adafruit_GPS_Library

These libraries should be included in order to run the Elcano_C6_Navigator code.

2.7 Limitations

- Our localization system cannot locate its location when GPS unavailable.
- Using the dead reckoning, know the position which is without GPS, but it is still not efficient.

2.8 Current Functionality

- Currently, this program gets bearing data from compass magnetometer LSM 303 and speedometer data from $C2$ Low level.
- Program calculates position based on Dead Reckoning (DR). DR is the process of calculating ones current position by using previously determined position.
- Fuzzy numbers algorithm is used to get the best estimated position based on the output from magnetometer plus speedometer and GPS position.
- The Navigator code sends Fuzzy output to C4. It also sends the GPS position to C2 Low level.
- All output is send to the SD card in a file named as LOG.csv. The file can be opened in a spreadsheet in order to analyze data and generate graphs.

2.9 Possible Extended Functionality

- Other sensor data like optical mouse to be investigated if that gives better accuracy for the position.
- Research on Pixhawk, if that can be used as an alternative for $C6\_Navigator$

3 Optical Mouse

3.1 Overview

Optical Mouse ADNS 3080 will be used to calculate the position of the robot using 2-D displacement vectors. The mouse sensor returns the average movement (in the x and y directions) of surface features that it sees. It does so by identifying texture in two subsequent frames and calculating the distance that the images have been displaced.

3.2 Connections

The connections may be different on different boards. This one is for Arduino Mega.

Table 1: Comparison of Parameters from DVG (directed) algorithm and DVG (undirected) algorithm

ADNS3080 optical mouse sensor	**Arduino Mega**
GND	GND
5V	5V
3V	Not connected (NC)
NCS	40 (any digital pin)
MISO	50 (MISO of arduino)
MOSI	51 (MOSI of arduino)
SCLK	52 (serial clock of arduino)
RST	44 (any digital pin)
NPD	NC
LED	NC

3.3 Experiments

3.3.1 Estimate the position of Optical mouse on Trike:

Since the laptop screen is properly illuminated, an experiment was conducted to study the variation of SQUAL (surface quality of screen) with distance from sensors 4.2 mm lens.

ADNS 3080 with 4mm lens is not suitable for land vehicles. It can be used at smaller range of distance like 4mm from road where 30*30 image can find texture. In the one to two foot range all the pixels look pretty much the same. When the pixels get larger, there is not enough texture in the road. 4mm lens was replaced by 12 mm lens. The setup was tested on the following road textures:

Results: It was found that the sensor gives maximum SQUAL value for 7 inch distance from surface.

3.3.2 Calculating speed at maximum and minimum frame rate of ADNS3080:

It is mentioned in the datasheet of ADNS3080 that maximum and minimum frame rate optical mouse are 2000 and 6469 frames per second respectively. The following image is used for calibration: The calculation is as follows:
Each square in the above image result takes 7*7 pixels area.

Total number of squares which can be placed in the 30*30 along length is 30/7 = 4.285.

Measurement of 4.285 squares in the checkerboard image is 19mm.

At minimum frame rate (2000 fps):

c1 frame will be captured in 1/2000 seconds. This implies 19 mm in length can be captured in 1/2000 seconds. Therefore the speed at this rate will be $19 * 2000/1000mps = 38$ metres per second.

At maximum frame rate (6469 fps):

1 frame will be captured in $1/6469$seconds. This implies, 19 mm in length can be captured in $1/6469$ seconds. Therefore the speed at this rate will be $19 * 6469/1000mps = 123$metres per second.

The above calculation is wrong since I have not considered overlapping between two images and optical mouse will not work for that.

For our required speed of 30kmph that is 8333 mm per second:

Let there is $x\%$ overlapping, so for every new frame we only get $y = (100 - x)\%$ of new frame area. 1 frame will be captured in$1/2000$ seconds (considering minimum frame rate). This implies $y\%$of $19mm$in length can be captured in $1/2000$ seconds. Therefore the speed at this rate will be$(y/100) * 19 * 2000 = 8333$mm per second. Solving the equation, we get y = 21.928 which implies x = 78.07105.

So from above calculation, we see that even at minimum frame rate the overlapping between two images is 78% which will be sufficient for our purpose.

3.3.3 Instructions

The optical mouse is to be mounted $7inch$ above the ground and the field of view should not be interfered by any obstacle. There should be proper lighting system to capture more features for $30*30$ image. Open "optical_mouse.ino" in arduino ide. Check the line 191 of the file. There can be following cases:

- **if 0:** This implies that the code will print two dimensional motion vectors along with SQUAL and intensity value.
- **if 1:** This code will grab frames and output them as matrix of intensities. The image can be seen using the python user interface "ADNS3080ImageGrabber". Open the terminal and go to the folder where you have saved "ADNS3080ImageGrabber.py". Run "python ADNS3080ImageGrabber.py" command in the terminal. The possible errors are: "could not open port /dev/ttyACM1" : In this case open "ADNS3080ImageGrabber.py" in any editor and change the line number 9 comPort = '/dev/ttyACM1' to the respective com port (shown in the right corner of arduino ide). Also, check the com port baud rate which is mentioned in line number 10. It should match with that of arduino. "Device or resource busy: '/dev/ttyACM0'": Please check if the serial monitor of arduino is open. Close it and run the code again.

3.3.4 Distance Calibration:

Distance can be calculated using the previous and present coordinates: $[(x1, y1)and(x2, y2)]$
Distance = $\sqrt{[(x2-x1)^2+(y2-y1)^2]}$
This distance needs to be calibrated in metric system units.
Note: If the SQUAL value is less than 10 then the value can be ignored.

4 Lane Detection

4.1 Overview

Lane detection algorithm will be used to calculate position of the robot from lane edges (mostly right lane edge). There are three parts of the algorithm as follows:

- **Perspective Transformation:** Once the position of camera is decided and camera is mounted, take an image of checkerboard like pattern to calibrate the camera.
- **Lane Detection:** This uses hough line transformation to detect lane edges after some preprocessing of image.
- **Communication between arduino and pi:** The distance calculated by rpi is sent to arduino. This data can be helpful in navigation.

4.2 Connections

Connect the Raspberry Pi Camera Board to the CAMERA slot of raspberry pi.
For communication between arduino and pi connect them using USB cable as shown below:

4.3 Perspective Transformation:

The relevant file is Vision/LaneDetection/transform.py and this is imported in our main code. We are using the four_point_transform function such that the first entry in the list is the top-left, the second entry is the top-right, the third is the bottom-right and the fourth is the bottom-left. This function is used to transform the image as seen for top. Either you can create an onclick event for mouse to click on the four corners of square or you can directly enter the coordinates of four corners in respective order since this has to be done only once. We have used the following image for reference. The four coordinates are $(coord = [(251, 314), (443, 306), (616, 435), (85, 445)])$ used in our main code. This will be again used in calibration.

4.4 Lane Detection:

The algorithm includes preprocessing of the image and some calculations. If we can get information about how wide the lane is then we can use this information to get a more efficient lane detection algorithm. The following calculation is unique to the camera and its position. For the above test image taken from Nikon camera, Calculation:
523 pixels on the above corresponds to 5 feet (60 inches)
The robot position is (523/2).
Therefore,
X = ((calculated x position of the lane - (523/2))*60) / 523 inches is the required distance.

5 Conclusions

This is the second chapter of the present dissertation. It is more interesting than the first one, for it is the last one.