3D Optical Flow User Interface and Tracking System

An Advanced Mechatronics ME6960 Final Project

Richard Kirby and Parker Conroy
# Table of Contents

1. Abstract
2. Project Overview  
   2.1. Dual Comparative Optical Flow System  
   2.2. Criteria for Success  
3. Hardware  
   3.1. CAD Model and Machined Components  
   3.2. Prototype Components  
      3.2.1. Design Metrics  
      3.2.2. Physical Components  
      3.2.3. Motor Selection  
   3.3. Electronics  
      3.3.1. dsPIC33E  
      3.3.2. Motor Drivers  
      3.3.3. Encoders  
4. Software  
   4.1. Diagram of Signal Flow and Wiring Diagram  
      4.1.1. Peripheral Connectivity  
   4.2. Control Loops  
   4.3. Embedded Software  
   4.4. Desktop Software  
5. Conclusion  
   5.1. Results  
   5.2. Future Work
Abstract

This paper details the design of a 3D optical flow based tracking system that serves as a platform for Natural User Interface (NUI) development and testing. In our design, a pair of optical flow sensors [1], which are capable of detecting object velocities in the real world X and Y coordinates (and calculating the distance to the object) are mounted in a custom dual gimbaled housing. Motion detected by the optical flow sensors is fed into a pair of PID controllers which in turn actuate the two gimbal motors to keep the tracked object in the center of the optical flow sensors’ view. The tracking system achieves millimeter accuracy while tracking objects at reasonable human velocities.

Project Overview

This project further develops the Dual Comparative Optical Flow sensor (DCOF) developed by Richard Kirby [2, 3]. Optical flow is a technique that measures the “apparent velocities of movement of brightness patterns in an image” [4] and produces velocity measurements in units of pixels/second. Conversion to real-world coordinates is not possible without knowing the distance between the optical center of the imaging optics and the object being tracked in the scene. The DCOF system (Figure 1) solves this problem by using two optical flow sensors, a mirror and a beam splitter that permit imaging of the same window in the 3D scene with both optical flow sensors along a partially collinear optical path. The two optical systems have different magnifications that produce different optical flow velocities in each of the two optical flow sensors. Depth is a function of the different magnifications of the two systems and the difference between the velocities detected by the two sensors (Equation 1). The key advantage of this arrangement over traditional stereo vision techniques is that there is no need to find corresponding points between the two images as the two images are of the same small window in 3D space. The key disadvantage of this arrangement is that by itself the system does not produce a dense depth map, but only finds delta X, delta Y, and the Z distance in a small observation window. By incorporating the DCOF sensor system into a two-axis gimbal mechanism, the system is capable of tracking a moving object in 3D space.
\[ Z = \frac{-b}{1-r_f r_c} \]  

(Eq. 1)

\( Z \) = depth  
\( b \) = baseline  
\( r_f \) = focal length ratio  
\( r_c \) = optical flow velocity ratio

In our system, the dual axis gimbal aims the DCOF optical system by rotating a mirror such that it aims the sensors’ x-axes and the entire DCOF assembly such that it aims the sensors’ y-axes (Figure two). These two rotational degrees of freedom create a workspace that can be modeled as the shell of a sphere about the mirror. The state of the system is updated at 500 Hz and sent over USB to a desktop computer that records the data, visualizes the path taken by the object, and allows the user to move the cursor of the computer by moving their hand.
The objectives for our project are:

1. Develop a gimbal mechanism capable of moving at 0.15 radians/second in two axes and accelerating at 1.5 rad/s/s, which permits tracking a human hand 1 meter from the optical center at 1 m/s under reasonable accelerations.
2. Collect optical flow data with one PIC controlled system, reliably transferring that data to a second dsPIC controlled system at 1000 Hz, and implementing two PID controllers on the dsPic to control two motors at 500Hz, thus enabling hand tracking.
3. Sending the full system state (six 8-bit DCOF parameters and two 16-bit motor encoder values) to a desktop via USB to allow real-time processing of motor position data and calculating depth from the optical flow data.
**Mechanical**

An exploded view of the DCOF PCB showing the optical flow sensors, optical assemblies, lenses, lens retainers, beam splitter, and mirror is shown in Figure 3 below. The optical system was custom designed with lens focal lengths of 75 mm and 100 mm. The optical system is capable of focusing between approximately 1 meter and infinity.

![Figure 3 DCOF PCB with Sensors, Optics, Mirror, and Beam splitter](image)

Figure 4 shows the CAD design of the assembled DCOF scanner. Figure 5 shows an exploded view of the mechanical components built and purchased for this project. The two circular end-plates were CNC machined and the drive shaft components were turned on a CNC lathe. The mirror assembly and other components shown in white were custom machined for the project. The y-axis motor is connected to the drive shaft via a helical flexible coupling. The entire assembly is mounted on an optical breadboard for stability.
The motors and motor drivers used in this project were chosen to meet the specifications shown in Table 1.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Quality</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angular velocity</td>
<td>0.15</td>
<td>radians/s</td>
</tr>
<tr>
<td>Angular Acceleration</td>
<td>1.5</td>
<td>radians/s/s</td>
</tr>
</tbody>
</table>

Table 1: Output Metrics

We initially attempted to use a pair of Minertia UGFMED-C9MRX11 motors without gearheads, however they were unable to generate the torque required to achieve the angular acceleration requirements. We refit the system with a pair of
Globe Motors with 922.31 gearheads which solved the torque/acceleration issue. An interesting side effect of using such a large gear reduction is the multiplicative effect on the encoders. We were using 500 count two-channel quadrature encoders which produce 2000 pulses per motor revolution. When multiplied by the gear ratio, we get over 1.8 million counts per mirror revolution. This required using 32-bit numbers for the encoder values and truncating the values (ignoring the last 9 bits) when using encoders in the calibration PID controllers because the encoder resolution was too great.

**Electronics**

**Overview**

Our application requires the dsPIC33E to interface with numerous external systems:

- Interface to DCOF’s dsPIC18F over framed SPI (dsPic in slave mode) to get optical flow data.

- Interface to desktop computer over USB for data logging and visualization of the position of the object being tracked.

- Communicate with the LCD via parallel interface and 3 GPIO lines.

- Read motor encoders via a logic level converter using the built-in QEI module.

- Drive a dual motor controller using two PWM outputs and 4 GPIO pins to control direction.

- Interface with four limit switches and three buttons using 7 GPIO lines.

In addition, the system uses 3 LEDs to indicate when (and which) limit switch are activated. By using the on-board switches as an interface, our system can be operated completely independent of the desktop computer. Table 2 shows the electrical components, the voltage at which they function, and the interfaces used.
<table>
<thead>
<tr>
<th>Component</th>
<th>Supply Voltage</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>dsPIC33E USB starter kit</td>
<td>3.3V</td>
<td>SPI Slave QEI Limit Switches (GPIO) Input Buttons (GPIO)</td>
<td>USB PWM Pulse PWM direction (GPIO) LEDs (GPIO) LCD (GPIO)</td>
</tr>
<tr>
<td>DCOF PCB with Pic 18F</td>
<td>3.3V</td>
<td>SPI</td>
<td>SPI USART</td>
</tr>
<tr>
<td>Logic Level Converter</td>
<td>5V/3.3V</td>
<td>Encoders (GPIO)</td>
<td></td>
</tr>
<tr>
<td>H-Bridge Motor Controller</td>
<td>3.3V/12V</td>
<td>PWM pulse at 3.3V PWM direction</td>
<td>PWM at 12V</td>
</tr>
<tr>
<td>Motor Encoders</td>
<td>5V</td>
<td></td>
<td>Two channel Quadrature</td>
</tr>
<tr>
<td>LCD</td>
<td>3.3V</td>
<td>GPIO</td>
<td></td>
</tr>
<tr>
<td>Desktop Computer</td>
<td>N/A</td>
<td>USB UART</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2: Major Electrical Assemblies**

**Electronic Components**

**dsPIC33E – USB Starter Kit**

The dsPIC33EP512MU8103E is a sixteen bit surface mount microcontroller with a large number of onboard peripherals including quadrature encoder inputs, pulse width modulation (PWM) outputs, analog to digital inputs, and an assortment of standard communication protocols. Specifically a dsPIC33E USB starter kit was used in this project, which incorporates the electronics required to program it via USB.

**Motors**

The motors used in our project are Globe Motor P/N 415A832 12V DC Gear Motors with a 922.31:1 gear ratio and Avago 500 count per revolution two-channel quadrature encoders.
Motor Driver

We used a Toshiba TB6612FNG dual motor driver on a Pololu breakout board. It takes two PWM CMOS level signals and four GPIO lines for direction and outputs two independent motor level signals for powering the two gimbal motors.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X, Y</td>
<td>12</td>
<td>0.80</td>
<td>12.3</td>
<td>5.6</td>
</tr>
</tbody>
</table>

Table 3: Motor Specifications

<table>
<thead>
<tr>
<th>Motor Driver rotation axis</th>
<th>Voltage Range [V]</th>
<th>Max Current per channel [A]</th>
<th>Max Speed [kHz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>X, Y</td>
<td>4.5 – 13.5</td>
<td>3.0</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 4: Motor Driver Specifications

Encoders

The Globe Motors come with Avago HEDS 5505 A4 two-channel 500 count per revolution quadrature encoders. The HEDs encoders require pull-ups on the outputs and a 5 volt supply.
Figure 6: Wiring Diagram
<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
<th>Peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>A14</td>
<td>SPI communication MOSI</td>
<td>DCOF</td>
</tr>
<tr>
<td>A2</td>
<td>SPI communication SCLK</td>
<td>DCOF</td>
</tr>
<tr>
<td>A15</td>
<td>SPI communication SS</td>
<td>DCOF</td>
</tr>
<tr>
<td>E0-7</td>
<td>LCD – data in</td>
<td>LCD</td>
</tr>
<tr>
<td>G15</td>
<td>LCD E</td>
<td>LCD</td>
</tr>
<tr>
<td>G12</td>
<td>LCD RS</td>
<td>LCD</td>
</tr>
<tr>
<td>G14</td>
<td>RW</td>
<td>LCD</td>
</tr>
<tr>
<td>D6</td>
<td>GPIO - input</td>
<td>Button 1</td>
</tr>
<tr>
<td>D7</td>
<td>GPIO - input</td>
<td>Button 2</td>
</tr>
<tr>
<td>D13</td>
<td>GPIO - input</td>
<td>Button 3</td>
</tr>
<tr>
<td>D0</td>
<td>GPIO - output</td>
<td>LED 1</td>
</tr>
<tr>
<td>D1</td>
<td>GPIO - output</td>
<td>LED 2</td>
</tr>
<tr>
<td>D2</td>
<td>GPIO - output</td>
<td>LED 3</td>
</tr>
<tr>
<td>C3/C4</td>
<td>PWM output</td>
<td>Motor Controller 1</td>
</tr>
<tr>
<td>C1/C2</td>
<td>PWM output</td>
<td>Motor Controller 2</td>
</tr>
<tr>
<td>D11</td>
<td>AIN1-PWM1 direction</td>
<td>Motor Controller 1</td>
</tr>
<tr>
<td>D10</td>
<td>AIN2-PWM1 direction</td>
<td>Motor Controller 1</td>
</tr>
<tr>
<td>D4</td>
<td>BIN1–PWM2 direction</td>
<td>Motor Controller 2</td>
</tr>
<tr>
<td>D5</td>
<td>BIN2–PWM2 direction</td>
<td>Motor Controller 2</td>
</tr>
<tr>
<td>I40/I41</td>
<td>Quadrature Encoder input</td>
<td>Encoder 1</td>
</tr>
<tr>
<td>I46/I47</td>
<td>Quadrature Encoder input</td>
<td>Encoder 2</td>
</tr>
<tr>
<td>USB</td>
<td>USB communication</td>
<td>Desktop</td>
</tr>
<tr>
<td>B10</td>
<td>Digital Input</td>
<td>Limit switch 1</td>
</tr>
<tr>
<td>B11</td>
<td>Digital Input</td>
<td>Limit switch 2</td>
</tr>
<tr>
<td>B12</td>
<td>Digital Input</td>
<td>Limit switch 3</td>
</tr>
<tr>
<td>B13</td>
<td>Digital Input</td>
<td>Limit switch 4</td>
</tr>
</tbody>
</table>

Table 5: dsPIC Pin Details
Software

Embedded Software Design

A large part of the project was invested in programming the dsPIC33E so that it could accomplish its tasks while running the 500 Hz control loop needed to update the two PWM outputs. All routines are non-blocking except the calibration routine, as we did not want the user attempting to update the PID gains while the dsPic was calibrating. Limit switches, which are interrupt driven via four input capture pins are disabled during the calibration process, as the calibration routine monitors the limit switches directly.

![Diagram of state machine and tasks]

**Figure 7: Overview of Embedded Software**

**main()**

The main routine runs a non-blocking state-machine (Figure 8) which allows the system to be completely controlled by the three buttons on the starter kit PCB including updating the PID gains and calibrating the gimbals. The state machine has 8 states as follows:

1. Display Quadrature Encoder Values and allows calibration
2. Display Kp for the mirror motor allow incrementing or decrementing value
3. Display Ki for the mirror motor allow incrementing or decrementing value
4. Display Kd for the mirror motor allow incrementing or decrementing value
5. Display Kp for the frame motor allow incrementing or decrementing value
6. Display Ki for the frame motor allow incrementing or decrementing value
7. Display Kd for the frame motor allow incrementing or decrementing value
8. Display X, Y, and image quality for each of the two optical sensors and allow calibration.

In addition to the state machine, main() calls the LCD service routine and USB service routine on each pass through the main while loop. The full machine state (optical flow values, PID cumulative error values, PWM output values, and encoder positions) is sent to the desktop via USB.

_IC1Interrupt(), _IC2Interrupt(), _IC3Interrupt(), _IC4Interrupt()

Four interrupt handlers for the four limit switches were implemented. When enabled, these interrupts set the PWM output to the respective motor to zero when a limit switch is actuated to prevent damage to the assembly. The limit switches are pulled high via 10K resistors so a zero on the associated input capture pin signals the switch has been actuated.

_SPI1Interrupt

The dsPIC was configured as a 16 bit framed SPI slave and the DCOF’s PIC18F bit-banged a 16 bit SPI master framed output. Four wire SPI was used because the PIC 18F uses the same SPI bus to sample the optical flow sensors. The PIC18F uses the upper byte to encode the type of data being sent (X, Y, and surface quality - SQUAL for each of the two sensors) and the lower byte contains the data. The PIC18F sends six 16-bit values every millisecond.

The _SPI1Interrupt extracts the upper byte and determines what type of data is contained in the lower byte. The lower byte is converted into a signed integer and used to update the cumulative value of the error signal into the PID controllers. When tracking a moving object with optical flow sensors, the output of the optical flow sensors is the incremental error. Cumulating incremental errors provides the output of the summing block in a PID controller, the error used by the proportional, integral, and derivative terms to calculate the required motor voltage which is then converted to a PWM signal.

The surface quality measure is used to ensure that the dsPic is getting valid optical flow data. Good surface quality is when the optical sensors are able to read over 50 trackable features. We set the dsPIC to only update the PID gains when we could see
at least 70 trackable features. We regularly see 120 trackable features when the system tracks a well lit object with one millimeter surface textures.

_T2Interrupt()

This interrupt handler is the heart of our code. It runs at 500 Hz and its primary purpose is to update the PID controllers. It takes the current values of the error, updates the integral and derivative terms in the PID algorithm and updates the direction pins and PWM signal values accordingly.

Timer two is also used to debounce the input switches. If an input switch stays closed for 20 consecutive passes through the T2 interrupt handler (40 mS), that routine signals a switch event which is responded to by the state machine in the main loop.

calibrationMotors()

This routine is called by the main routine via the user interface. It runs the mirror motor at half speed in the CW direction until the limit switch is activated. At this point it resets the quadrature encoder values to zero. It then runs the mirror motor at half speed in the CCW direction until the second limit switch is activated. It computes the encoder counts from limit to limit, takes half the encoder counts, and uses that value and a PID controller to center the mirror. The calibrateMotors() routine then follows the same sequence for the second motor.

This routine uses blocking code intentionally to prevent the user from accessing the user interface during the calibration procedure. However, all the interrupts still fire, so code is incorporated in the input capture interrupt handlers as well as in the timer two interrupt handler to prevent conflict between the PID controllers that use the output of the optical flow sensors and the calibration PID controllers which use the output of the motor encoders.

IcdService LCD()

This routine is called each time through the main while loop. It uses a case-switch structure to update the LCD based on the state-machine state.

Desktop Software Details

A Linux desktop application was developed to visualize the velocity data and act as the user interface. This program works in the Robot Operating System (ROS) shell while taking advantage of the open source point cloud library (PCL) for visualization. Videos of these functions can be found at
Conclusions and next steps

The system is capable of tracking reasonable speed hand movements in 3D space (http://www.youtube.com/watch?v=X25jsmYaiR8&list=HL1354932183&index=1). The tracking of hand movements in the x-direction is relatively impressive, fast, and without a detectable delay. However, there is a slight, but detectable delay in the y-direction due to the reduced gains on the y-axis PID controller that were required to prevent oscillation. We believe the oscillations we experienced in the y-direction with higher PID gains were due to the low natural frequency of the y-axis assembly, which in turn was due to the flexible coupling between the y-axis motor and the assembly.

System operation could be improved by using a more rigid coupling between the y-axis gimbal motor and the gimbal mechanism. We purchased pinion gears to replace the flexible coupling, and are in process of making this modification.

The combination of optical flow sensors and optics chosen require a reasonable quantity of light to image well. With a fixed lighting system, the entire working area
has to be illuminated. However, if a collimated light source were mounted on the gimbal mechanism, we would only need to illuminate the small area (less than 1-inch square) that is being imaged. This substantially reduces the amount of illumination required and allows the system to work in most environments without external illumination.

While the optical system is capable of focusing between 1m and infinity, the illumination system and the target pattern do not allow taking advantage of larger depths of field. More work is required to improve the depth of field to truly take advantage of 3D hand gestures.

References