Implementing a gesture controlled pan and tilt CCTV camera system on the ATMEGA32A


I. EXECUTIVE SUMMARY

THROUGH the use of the ATMEGA32A (as part of the OUSB Board development platform [1]) it is possible to implement a low-cost gesture-controlled camera movement system with a high level of functionality and minimal shortcomings. The functional subsystems achieved through this project include: dual servo motor control with velocity and easing, simple gesture and proximity detection and live video monitoring of the CCTV camera’s visual feed. The accelerated timeline for development limited the variety of potential sensors and components that were tested for use and the depth and precision of the final project’s functionality.

II. INTRODUCTION

This report is focused on the implementation of a camera system capable of pan and tilt functionality through the use of two servo motors and a commercially available gesture sensor (Broadcom APDS-9960 [2]), on the ATMEGA32A microcontroller. Implementation will take place in Embedded C and assembler, these two methods were compared and contrasted with regard to speed, size of compiled code and functionality. The project scope covers the design of the system, sourcing of components and the development of all code by the author (no external libraries or functions not built into the language or compiler have been used).

RMIT University, Melbourne, AU 2018.

III. ESSENTIAL BACKGROUND INFORMATION

Pan, tilt and zoom cameras (PTZ cameras) have long been the standard in high end security systems and robotics research dating back to the 1970s. In recent times, these highly flexible camera systems have also found their way into home security, conference centres, lecture theatres and the web based television industry. A significant barrier to entry for these systems is the high cost and often complex interface protocol. The aim of the project was to approximate the functionality of high end PTZ cameras at a lower price and with fewer external components by leveraging the built-in hardware features of the ATMEGA32A. Specifically this microcontroller’s ability to generate highly accurate pulse width from the 16bit timer. In this project, the simple interface of a joystick (common to many PTZ camera systems) has been replaced with the novel use of a gesture sensor, as the use of such a sensor was a requirement of the project.

IV. TECHNICAL WORK AND RESULTS

The goals of this project were to create a gesture-controlled pan, tilt camera system that is flexible, simple and user friendly.

Flexible: the system will aim to be as modular as possible not relying on components that are not easily sourced or code that is not owned by the creator or freely available in the component’s data sheet.

Simple: the system will aim to use minimal external components to achieve its goals, relying on the ATMEGA32A’s built-in hardware and software functions as much as possible.

User friendly: the system will aim to be simple and intuitive to use, it will not be possible to break or damage the system through standard use.

V. FUNCTIONAL BLOCK DIAGRAM

![Functional block diagram for system.](image)
VI. IMPLEMENTATION

The implementation of this project was carried out in the following order:

**Servo motor control in Embedded C**
- Servo motor timings and range characterised on Keysight DSOX3014A using the built-in waveform generator.
- Two 1-1.7ms variable pulse width signals at 50Hz generated on ATMEGA32A using Timer1 (a 16bit timer with dual output compare registers: OCR1A and OCR1B).
- Timer1 output validated against results from Keysight DSOX3014A.
- Generate a number of high-level functions to make servo control more streamlined, including: servo position conversion from clock ticks to degrees and vice versa, transition from one position to another with velocity control, simple acceleration and software limits on servo angle.

**I2C gesture sensor integration in Embedded C**
- Establish I2C communication with sensor.
- Retrieve manufacturer ID and compare to data sheet to confirm successful communication.
- Create list of setup registers, read/write to them and confirm success using the PORTB LEDs as a debug (OUSB Board lacks built-in debugging or serial terminal).
- Step through the gesture reading process register by register comparing readings with expectations.

**Power supply creation and testing**
- Test Meanwell power supply with oscilloscope and observe start-up and shut-down behaviour.
- Design and construct a simple 12V DC to 5V DC regulator using off-the-shelf adjustable buck regulator/stepdown converter.
- Add additional components to improve power supply performance including diodes for reverse polarity protection, capacitors for power supply filtering and resettable polly fuses for over current protection.

**Servo motor control and I2C sensor integration**
- Servo control and sensor integration code combined into a single project.
- Output from gesture sensor code translated to servo control code.
- Tested and validated via observation.

**Working code translated from Embedded C to Assembly**
- Servo control code written in Assembly.
- Servo code tested and validated with real world hardware.
- I2C sensor integration written in Assembly.
- I2C data tested and validated with real-world hardware.

**Testing and code refinement**
- Components mounted on plywood board.
- System tested in-situ, code tweaked and refined to better respond to real-world conditions.

**Validation and reflection**
- System compared with initial system design and strengths/weaknesses considered against initial design as well as the professional systems this project is emulating.

VII. COMPONENT BLOCK DIAGRAM

**Servo motors and pan/tilt bracket**: Two Hitech (HS485HB) servo motors mounted as a pan and tilt bracket using an SPT100 Pan and Tilt Kit from Servo City [3].

**Camera**: A Watec 902A mini black and white CCD camera with CS mount lens, requiring 12V DC power [4].

**Screen**: A 3.5” TFT screen produced by Good Display [5] with a composite video input, requiring 12V DC power.

**OUSB Board**: The OUSB board [1] is a development platform for the ATMEGA32A [6] designed for use by RMIT electrical engineering undergraduate students. This board breaks out many of the major pins to standard development board peripherals such as, LEDs, DIP switches, RS232, potentiometer and a light dependant resistor. The OUSB board forms the core of this project although this project does not use any features that could not be replicated on another ATMEGA32A platform.

**Gesture Sensor**: A Broadcom APDS-9960 placed on a development board by Aidafruit [7]. This board features the gesture sensor as well as level shifting logic, pull-up resistors and a 5V to 3.3V power regulator. The APDS-9960 is capable of detecting hand gestures as well as proximity and light colour using an infrared LED mounted on the top and four inferred sensors mounted on the four cardinal points of the sensor face. Gesture detection is performed by a built-in state machine whose data is presented in a FIFO buffer.

**Power Supply**: With flexibility in mind, a Meanwell 110V/220V to 12V AC/DC supply (LRS-35-12) was used to generate the 12V DC required for the Watec camera, the Good Display TFT screen and the 5V DC regulator which, in turn, powers the servo motors. As both the Meanwell and
DC regulator for the servos are switch mode power supplies, extra care needed to be taken to ensure that the fast voltage spikes often generated by this style of power supply were not passed on to the components within the system. This was achieved using diodes for reverse polarity protection, electrolytic capacitors for smoothing voltage spikes and ferrite beads to reduce high frequency noise.

![Power supply block diagram.](image)

Resettable polyfuses were also used to protect the 12V and 5V supplies from overdrawing current in the event of malfunction or fault.

**VIII. TECHNICAL CONTENT**

From the results in Table 1, and by examining the code for each implementation, it is clear there is a trade-off between compiled size, readability and running speed.

- **Compiled size**: The Embedded C code implementation represents a 587.5 percent increase in size verses the Assembly code implementation, although not a lot of credence should be given to these exact figures as the Embedded C code implementation has significant increased functionality over the Assembly code implementation.

- **Readability**: Many of the conventions of Embedded C code by their very nature create a highly readable document. For example, function prototypes create a central location to view all the documents functions, argument types and comments in a central location, whilst in Assembly this would need to be hand written by the programmer as a comment and would not reflect changes in the actual code in the way that function prototypes do in Embedded C.

- **Speed**: Less lines of compiled code in almost all cases correspond to faster execution time. Whilst in Assembly the programmer may be able to execute a port write or loop in two to three lines of code, there is the additional mental burden of remembering the uses and expected contents of registers as well as the additional effort of working with variables that do not align with the bus width of the processor architecture. In Embedded C many of these burdens are alleviated by the compiler.

**IX. DISCUSSION AND CONCLUSION**

The size and complexity of this project was deliberately selected to push my learning as far as possible in a relatively short time frame. I found myself feeling unchallenged by the list of projects presented in the major project brief and arranged a meeting with the course coordinator to pitch a project that would build on and extend the knowledge I had gathered during the semester. Going into this project there were a significant number of unknowns and gaps in my knowledge regarding the use of timers, waveform generation and the I2C protocol.

My first major hurdle took the form of getting a working I2C signal from the hardware I2C pins on the ATMEGA32A. After writing my own I2C driver using the data sheet as a reference there was no signal on the SDA and SCL pins. The driver I had written appeared to work in simulation and the pins were able to be flipped by writing directly to the port making the problem appear to be a software issue. My next problem solving stop was to upload example code for I2C supplied by the lecturer and again, no output signal. I then consulted with the designer of the OUSB board to see if there might be any kind of conflict preventing me from using the hardware I2C, there was no known issues.

At this point, the issue of being unable to get a working I2C bus up and running looked set to derail my already tight timeline. I turned to researching and writing a software I2C driver, which despite being a huge amount of work, was able to be checked and validated on an oscilloscope every step of the way.

**TABLE 1**

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Embedded C</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>816* (2952 disassembly)</td>
<td>759*</td>
</tr>
<tr>
<td>Functions/sub routines</td>
<td>19</td>
<td>28</td>
</tr>
<tr>
<td>Project size (compiled)</td>
<td>5912 bytes</td>
<td>860 bytes</td>
</tr>
</tbody>
</table>

*includes whitespace used for readability
development. The setback of the hardware I2C issues ended up taking around a week of work to overcome. However, after successfully retrieving the device ID from the gesture sensor using software I2C late one night, the project was back on track. From here, setting up the gesture sensor’s registers and successfully retrieving and processing the FIFO gesture data was a process of trial and error involving comparison of the retrieved register contents (displayed on the PORTB LEDs) against the expected results from the data sheet. Realising the task of porting my software I2C driver from Embedded C to Assembly would be a very challenging task I once again attempted to implement a hardware I2C solution. Unfortunately, I remained unsuccessful and was forced to move ahead with the time consuming task of porting the software solution to Assembly. Once completed, a separate working servo PWM project was generated before the two were combined, and teething problems resulting from the two blocks of code being combined were debugged and resolved.

The final product is fully functional and a success in terms of the goals of the project, however due to the tight time frame and setback involving the I2C hardware driver, some features were unable to be implemented with a level of quality and usability I would be fully satisfied with. With additional time, improvements could be made to increase the reliability of the gesture detector and allow for faster more fluid gestures.

X. APPENDIX

Equations for translating between servo position in degrees and OCR values (clock ticks) in the TIMER1 registers.

\[
\frac{(\text{Input Value in Degrees})}{360} \times (301.8867925) + 188.6792453
\]

In the specific example of translating between degrees and clock ticks, the following formula was used,

\[
\frac{(\text{Input Value in Degrees})}{360} \times (301.8867925) + 188.6792453
\]

The values for output range and output range minimum were calculated by taking the desired length in milliseconds (minimum 1ms, maximum 2.6ms) multiplying this value by 1000 and then dividing by the timer’s clock period at 50Hz (5.3us).

In the Assembly implementation a look-up table of clock ticks was generated to avoid the floating point math of the previous equations.

The equation used to generate the delay times used in the ease out function consists of the cubic below.

\[
\frac{(\text{CurrentPos} – \text{EndPos})^3}{(\text{StartPos} – \text{EndPos})130}
\]

REFERENCES


