UNIVERSIDAD DE MONTERREY

División de Ingeniería y Tecnologías

Departamento de Inegniería

ACTIVITIES REPORT

Final Evaluation Project

Dr. Jorge de Jesús Lozoya Santos
Project Assessor

Pedro Aguiar
Mechatronics Engineering Student

May 16th, 2014.

Contents

List of Figures

List of Tables

1 MEMORANDUM 01

April 23 - Mayo 2

1.1 Proceso Administrativo

En esta semana todavía hubo pendientes administrativos que se reportan a continuación, como referencia e información para futuros estudiantes de la UDEM que vayan a participar en un programa similar. Al llegar al laboratorio lo primero que hiceron fue llevarme con Jennyfer DUBERVILLE para revisar los procedimientos administrativos. Se revisó que la papelería que había sido enviada por internet estuviera correcta. La papelería consistió en:

- Credencial de estudiante
- Copia del último certificado (en mi caso el de la preparatoria)
- Copia del pasaporte
- Un contrato (firmado por mí y por el director de DIT, por parte de la UDEM)

Después de corroborar la papelería se le entregó un formulario a mi asesor y a mí me dieron un reglamento acerca del uso de los recursos del laboratorio (internet, equipo electrónico, etcétera). Se me hizo firmar un papel en el que decía la fecha de llegada al laboratorio y donde declaraba haber leído el reglamento. Se me recordó que los primeros días de Mayo tendría que pagar las cuotas de "social security" y "civil liability". También se me informó que pare recibir la gratificación tengo que abrir una cuenta en un banco Francés y me dijeron que me iban a contactar con una persona que podía ayudarme con eso.
Por último se me proporcionaron las credenciales de acceso a Internet y mi computadora. Se instaló el equipo de cómputo que se me asignó y se me explicó cómo utilizarlo, así como algunos comentarios finales (no se puede conectar nada a la red alámbrica, no puedo instalar software que no esté en los repositorios de Debian, cómo acceder a mi espacio en el servidor de la escuela que está respaldado, detalles de ese tipo).

1.2 Actividades

1.2.1 PCB comunicación entre cámara y raspberry pi.

Durante el primer día de trabajo se hizo un bill of materials (Tabla 1.1) para hacer una tarjeta de circuito impreso que conecte una cámara OV7670 con el raspberry, el objetivo siendo que el Carrita tenga capacidades de visión computarizada. La diferencia con la de la UDEM es que Varrier externó que le gustaría recibir en el raspberry:

- Máxima resolución (640x480).
- Máximo framerate (30fps).

Componente	Cant.	Precio Unitario	Costo (euros)
PIC18F45K80-I/P	1	3.46	3.46
Capacitor cerámico 0.1 uF	10	0.386	3.86
Capacitor cerámico 1 nF	25	0.114	2.85
Capacitor tantalum 10 uF	5	0.822	4.11
Capacitor cerámico 15 pF	10	0.262	2.62
Resistor 470 ohm	10	0.034	0.34
Resistor 10k	10	0.025	0.25
Resistor 330	10	0.034	0.34
Resistor 470	10	0.034	0.34
Cable (jumper terminals)	30	0	0
Headers (pins)	2	3.43	6.86
Quartz	1	2.11	2.11
Switch SPDT	3	2.27	6.81
PicKit2	1	33.99	33.99
		Subtotal	67.94
		Total	81.528

Table 1.1: Bill of materials para el PCB de conexión entre el PIC y la cámara.

- Imagen a color.

Observaciones sobre el bill of materials:

- Costo elevado
  - 40.74 euros de componentes de una placa.
  - 33.99 euros para el programador.
- El proveedor impone cantidades mínimas de componentes.
- No está considerado el costo de mandar a hacer la placa.
- No se podría utilizar el microcontrolador de la UDEM porque no estaban considerados los bits para el color.
- Por lo tanto, el PCB tendría que ser diseñado completamente desde cero con la sensación (personal) de menos herramientas y proveedores menos accesibles.

1.2.2 Alternativa a Microchip.

Debido a lo costoso en dinero (componentes) y tiempo (diseño y armado) que resultó ser el diseño de un PCB para establecer la comunicación con las cámaras procedí a buscar una alternativa que ofrecerle a Varrier. La mejor alternativa fue utilizar un STM32, porque:

- Tienen Digital Camera Module Interface (DCMI).

- Tienen Direct Memory Access (DMA).
- Combinando DCMI+DMA se pueden guardar frames en memoria usando sólo hardware (el procesador queda libre para usarlo para otras aplicaciones).
- Hay tarjetas de evaluación de bajo costo (una de 12 y otra de 21 euros).
- Programación por USB.
- Varrier tiene experiencia y confianza en estos dispositivos.

Se investigó para validar que las tarjetas de evaluación pudieran hacer la comunicación con la cámara. También se revisó si se podrían utilizar los sensores y la pantalla LCD de la tarjeta (si no interferían con los pines que se necesitaban para la cámara). Se tomó la decisión de comprar dos tarjetas 32F429IDISCOVERY.

1.2.3 Estudio e implementación en C de transformada de Hough.

El código implementado en la UDEM para detectar la línea que se está siguiendo utiliza OpenCV, una librería que tiene una gran variedad de algoritmos para hacer visión computarizada. Es un objetivo del proyecto que el Carrita pueda medir la distancia que hay hacia un vehículo que esté enfrente y esta función no estaba implementada.
Antes de comenzar la implementación decidí deshacerme de la dependencia de OpenCV y desarrollar código en C para una transformada de Hough basado en los siguientes hechos:

- OpenCV es una librería muy generalizada con una gran cantidad de funciones (la descarga de la versión estable actual, 2.4.9 es de más de 87MB para Linux).
- Para la detección de línea sólo se estaban usando dos "funciones complejas" de OpenCV.
  - Canny edge detector.
  - Hough transform.
  - Seguramente menos del 0.5% de los 87MB.
- La transformada de Hough de OpenCV no puede usarse para detectar al vehículo de enfrente (sólo se detectan líneas rectas y círculos).
- OpenCV hace el cálculo completo de la transformada en cada frame. Nuestra aplicación puede aprovechar el hecho de saber que es un video para no realizar el cálculo completo en cada frame y buscar alrededor del punto conocido del cálculo anterior.

Además tenemos las ventajas típicas de programar una versión independiente de librerías:

- Conocimiento profundo de la transformada (funcionalidad menos abstracta).

- Adaptación específica a la necesidad (minimalista, rápida, a la medida).
- Libertad en funcionalidad, optimizaciones e interfaces.

En algún momento se iba a llegar al límite de la capacidad del raspberry utilizando OpenCV y para empujar ese límite se tendría que bajar a una solución específica sin uso de librerías generalizadas.
Para implementar en C la transformada de Hough se llevaron a cabo las siguientes tareas:

- Estudio de la teoría detrás de la transformada de Hough.
- Búsqueda y comparación de código existente para la transformada.
- Adaptar una versión a usar sólo C (sustituir librerías).
- Implementar optimizaciones
  - (Ab)uso del preprocesador (para reducir bifurcaciones).
  - Aritmética de punto fijo.
  - Tablas de funciones trigonométricas.
  - Memoria asignada estáticamente.
  - Region de interés.
  - Reducción de grados de libertad.

Se utilizó SDL para dibujar en pantalla (no es necesario, pero sirve para ver y comprobar resultados) y se adaptó fast-edge (http://goo.gl/ARWJZj), un proyecto encontrado en internet, para hacer detección de bordes.

1.2.4 Revisión bibliográfica de aplicaciones de la transformada de Hough.

Se han descargado 25 artículos relevantes al tema en los que se están observando aplicaciones de extracción de características usando la transformada de Hough. Al seleccionarlos se observó que hablan de las limitantes de la transformada, pequeñas modificaciones para resolverlas, así como propuestas y resultados experimentales.

1.3 Resultados

1.3.1 PCB comunicación entre cámara y raspberry pi.

- Bill of materials de la tabla 1.1.
- Debido al costo elevado en tiempo y dinero se decidió buscar una alternativa.

1.3.2 Alternativa a Microchip.

- Se utilizará un STM32.
- La tarjeta a utilizar es la 32F429IDISCOVERY.

1.3.3 Estudio e implementación en C de transformada de Hough.

- Se entendió toda la teoría detrás de la transformada.
- Se implementó la funcionalidad de la transformada de Hough tradicional utilizando sólo C.

1.3.4 Revisión bibliográfica de aplicaciones de la transformada de Hough.

- Empecé una revisión bibliográfica
- Observé las modificaciones (y su justificación) que se hacen a la transformada de Hough "tradicional" para implementarla en aplicaciones reales.

1.4 Pendientes

- **Gantt:** Le puedo adelantar que Varrier y yo estuvimos de acuerdo en que en dos meses tenemos que haber hecho el control de distancia (y de ser posible también lateral) utilizando las cámaras y haber envíado un artículo a alguna conferencia. Después de esos dos meses me dijo que lo decidiriamos después en base a mis intereses, etcétera. La única opción que le ofrecí fue la de simular la flotilla de vehículos y le pareció buena idea pero me dijo que lo siguiera pensando y cuando estuviera la fecha más cercana tomábamos la decisión. Entiendo algo como que el plan de colaborar con otro equipo (que se iba a encargar de la simulación de la flotilla) se vino abajo.
- **Compras:** La próxima semana estarán llegando los componentes.
- **Repositorio y Dropbox:** Estuve buscando una forma de sincronizar y respaldar el trabajo, ya me rendí y parece que no habrá formas cómodas (por restricciones de no instalar software que no esté en los repositorios de Debian y debido a que estamos atrás de un proxy que necesita contraseñas), pero aun así subiré todo el trabajo a algún lugar, le aviso la decisión en el próximo reporte.
- **Las actividades se dibujan así:** programación de STM32, trabajo electrónico para envíar las imágenes de la cámara al raspberry, desarrollo de transformada de hough específica para la aplicación, sacar modelos adecuados para la aplicación, implementar control tolerante a fallas.

2 MEMORANDUM 02

May 5 - May 9

2.1 Actividades

2.1.1 Revisión bibliográfica de aplicaciones de la transformada de Hough.

<Tabla pendiente>

2.1.2 Implementación en C de la Transformada Probabilística de Hough.

Bajo las mismas justificaciones por las que se decidió implementar en C la Transformada de Hough y, además, como resultado de los puntos expuestos en la revisión bibliográfica se decidió que se iba a desarrollar la Transformada Probabilística de Hough. Esta transformada no es matemáticamente una transformada porque, como dice su autor original en el paper donde la persentó, no se puede revertir.

Figure 2.1: Imagen Torre Eiffel original en blanco y negro.

- Variante conceptual de la Transformada de Hough que resuelve (o atenúa) sus desventajas más importantes.
- OpenCV cuenta con la implementación de ambas: la Transformada de Hough y la Transformada Probabilística de Hough (más nueva y más usada).

Con el objetivo de contextualizar se presenta una comparación del procesamiento sobre la misma imagen (Fig. 2.1) por ambos algoritmos para comparar los resultados. El resultado gráfico al utilizar el algoritmo tradicional se muestra en la figura 2.2, mientras que la salida al utilizar la Transformada Probabilística se muetra en la figura 2.3. La prueba consistió en procesar la imagen por ocho mil iteraciones, para evitar ruido de programas externos y ajenos al experimento. Los programas no se ejecutaron al mismo tiempo y había la misma cantidad de programas corriendo en la computadora cuando se hicieron las pruebas.

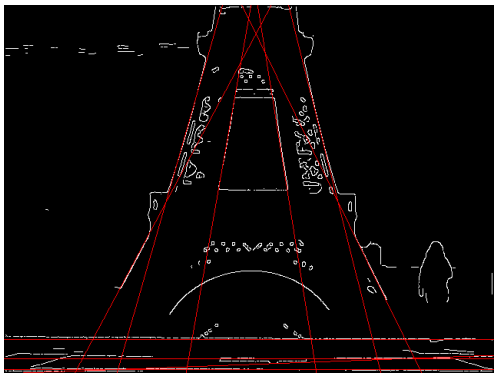

Figure 2.2: Imagen Torre Eiffel procesada con la Transformada de Hough.

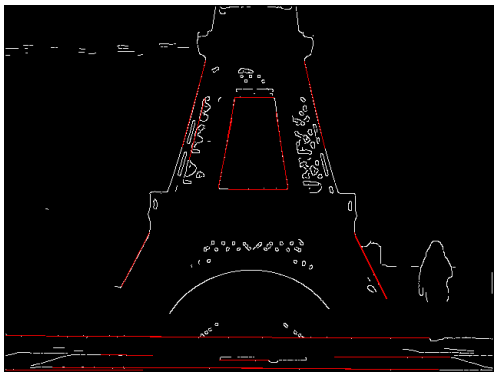

Figure 2.3: Imagen Torre Eiffel procesada con la T. Probabilística de Hough.

Se pueden hacer las siguientes observaciones en cuanto al nivel de similitud de los resultados:

- En la versión tradicional no se está computando los puntos que delimitan la línea (por eso atraviesan toda la imagen).
- La versión probabilística detectó todas (y más) las líneas que detectó su contraparte.
- Comparando el tiempo que les tomó realizar la prueba se obtuvieron los resultados de la tabla 2.1.
- El tiempo es despreciablemente menor para la Transformada Probabilística. Pero además esta transformada hace más cosas que la tradicional. Entonces hace más en menos tiempo.

Característica	Tradicional	Probabilística
Tiempo total (s)	163.97	154.52
Tiempo por frame (ms)	20.5	19.32
Frecuencia (Hz)	48.79	51.77

Table 2.1: Comparación de ambos algorítmos al procesar la imagen 8000 veces.

- Los parámetros de búsqueda de líneas de la Transformada Tradicional estaban prácticamente "sintonizados" y probados.
- Simplemente cambiando los parámetros de la Transformada Probabilística se pueden obtener mejores resultados (de por sí son buenos).
- Esto es, la transformada Tradicional se muestra funcionando al límite, mientras que la Probabilística todavía se puede mejorar.

2.1.3 Planeación.

La planeación del proyecto no se había hecho porque los objetivos no estaban bien definidos. Ahora que se definieron un poco mejor los objetivos era necesario separar el objetivo en tareas y esas tareas en subtareas para tener una percepción gráfica de la cantidad de tiempo requerirá el proyecto.
Se separó el objetivo en las siguientes secciones principales:

- **STM32:** Código de adquisición y envío de imágenes.
- **Físico:** Hardware, montaje de cámaras en el vehículo, etcétera.
- **Raspberry:** Código de recepción de imágenes y gestión de tareas (prioridades, scheduler, etcétera).
- **Visión:** Procesamiento de imágenes.
- **Control:** Extracción de modelos, sintetización de control.
- **Paper:** Investigación necesaria, documentación y publicación de los resultados del proyecto.

2.2 Resultados

2.2.1 Revisión bibliográfica de aplicaciones de la transformada de Hough.

- Se detectaron las principales debilidades de la transformada de Hough.
- Se vieron varias soluciones que se han propuesto para eficientizar la implementación.
- Se decidió implementar el código en C para la Transformada Probabilística de Hough.

2.2.2 Implementación en C de la Transformada Probabilística de Hough.

- Se implementó satisfacoriamente.
- Es una variación del concepto original que hace más en menos tiempo.
- Sigue hacer más optimizaciones al código y generalizarla para detectar vehículos.

2.2.3 Exploración de proyectos de STM32.

- Se recolectaron programas de ejemplo de las funciones que se usarán (DCMI, DMA, SPI).
- Está planeado empezar a compilar estos proyectos la próxima semana.

2.2.4 Planeación.

- Se definieron actividades a realizar para el objetivo.
- Se asignó una cantidad de tiempo aproximadoa a cada actividad en forma de Gantt (Fig. 5.1).

2.3 Pending

- **Diseño de la estructura mecánica para montar la cámara:** Pensaba tenerlo para esta semana, sin embargo en el Gantt (Fig. 5.1) ya lo puse para la siguiente.
- **Compras:** Se esperaba recibir lo que se compró durante esta semana pero no fue así.
- **Benchmarking/profiling:** Se asignó un tiempo a esta actividad en el Gantt también.

3 MEMORANDUM 03

May 12 - May 16

3.1 Activities

3.1.1 Camera mount design

- The electronic components were received this Monday.
- Measurements of the size of the camera and its PCB components were taken.
- After observing Carrita's front structure the only readily available mounting point was identified as a thin plate with four screw holes.
- Measurements of this thin plate were taken.
- A camera mount was designed aiming at:
  - Centered cameras.
  - Aligned cameras.
  - Fixed, constant distance between cameras.
  - Keeping a DOF (angle/inclination of view towards the floor).
- CAD software was used to draft mechanisms and spot possible conflicts.
- After solving some design conflicts found the original concept part and drawings files were generated.

3.1.2 Edge detection research

As mentioned in previous sections, the code used used for edge detection is from FAST-EDGE project. Some modifications were done to the arguments of the functions, the file loading code was removed and the functions were used to work directly on memory.
The code does mainly two things:

- Gaussian blurs the image.
- Detectes and draws edges (white) over a black background.

These two processes' input and output were well known by me but their internals were not. I studied these topics to know which optimizations could be done to the code and which lacking functionality could be added.
The following conclussions can be made after studying the code and the theory behind it:

- Region of interest (ROI) can be added to the code to avoid running the function on the whole image if it is not necessary.

- Functions are not inlined. This does not guarantee the compiler is not inlining but it is a potential boost.
- Fixed-point arithmetic is not used either.

The most useful and interesting finding of this research was that the code looked like it could be implemented on a GPU. This topic is presented in the next section.

3.1.3 General-purpose computing on graphics processing units

Graphical processing units are highly parallelized processors used (mostly) for graphics. Central processing units may have multiple cores but they are far of being as powerful as a GPU when it comes to solving computations which can be parallelized. While reading the code behind edge detection I remembered GPGPU, which is using the GPU for computations typically done by a CPU (i.e. not graphics).
Some GPU vendors have introduced or are introducing interfaces to freely use the GPU. The main open project for this is called OpenCL. For processors without OpenCL support, GPGPU computation has been done using OpenGL shaders to perform graphics-unrelated computations by mapping them to a graphics space (performing calculations of fast fourier transform and other digital signal processing operations much faster than on the host CPU).
Some (the most important) research results for related GPU published work are:

- OpenCL is not supported on the Raspberry Pi.
- Raspberry Pi's OpenGL implementation is a complete OpenGL ES 2.0.
- This is a video[1] showing Pi's GPU computing and displaying sixteen textures.
  - One of them is a Gaussian filter.
  - All are drawn at the same time in real time.
  - Output is sent out through HDMI 1080p.
  - OpenGLSL code is available.
- Markéta Dubská et al.[2] show a GPU implementation of line detection using OpenGLSL.
  - The GPU implementation outperforms OpenCV running on all the cores of an i7-920 processor.
  - Code not available.
- I have experience using OpenGL for graphics but not for computations.
- I began to study OpenGLSL because its an area worth of exploration.
  - It has space for research and inovation.
  - Impact on any device with an OpenGL ES 2.0 capable video card.
  - This includes most smartphones

  - Previous experience with OpenGL makes spending time on this less costly.
  - On full success this will boost our possibilities significantly.
  - Otherwise it will still help a bit.

3.1.4 STM32 - IDE installation

- IDE options exploration.
- Two were installed and tested on Windows based on internet resources.
  - CrossWorks for ARM.
  - IAR Embedded Workbench ARM v7.10.
- IAR Embedded Workbench will be used for the project.

3.1.5 STM32 - SPI and DMA

- A communication test with the Raspberry Pi was performed, confirming the following keypoints work:
  - IDE.
  - Programmer.
  - Debugger.
  - SPI (slave mode) and DMA (memory to peripheral).

3.1.6 Raspberry Pi - SPI

The most common (and documented) way of accessing the SPI peripheral on the Raspberry Pi is through the Linux kernel module (whether accessed from python, shell, C). I knew it was not the fastest option but I did not know how the fastest one work. After the corresponding research and tests it can be noted that:

- Kernel's SPI module does not allow "full freedom" access to the chip peripheral.
- This limits the transfer speed.
- bcm2835 library allows low-level access using high-level programming.
- The library was compiled and installed.
- A communication test with the STM32 was successfully done.

3.1.7 Raspberry Pi - Performance improvements

- **GPU.** Already explained at another section.
- **Bare bones.** After finding the Linux is not needed for SPI usage the question about whether it was needed for anything arose and it turned out to be possible to run ARM code directly from the SD card. Work has been done on that. This is just something to have in mind because there are some very useful Linux features that are still needed (drivers, process management, pipelines).
- **Overclocking.** An official maximum overclock was performed, this sets the Raspberry Pi clock at 1 GHz (42% faster than without overclock).
- **Shutdown unused modules.** Some Linux modules can be turned off to save CPU usage. That was known but an interesting finding was that the HDMI output can be turned off (which is a relatively heavy process that will not be needed). This is something to have in mind: turning down as many things as possible.

3.2 Results

3.2.1 Camera mount design

- Requirements definition.
- CAD part and drawing design. See figures 5.2, 5.3 and 5.4 at the Appendix.
- Research on the camera's positioning for propper stere visions.

3.2.2 Edge detection research

- Knowledge about gaussian, sobel and canny theory used for edge detection.
- Possible optimizations to and limitations of the code were identified.
- Hint about GPU usage.

3.2.3 General-purpose computing on graphics processing units

- Knowledge about Pi's GPU.
- Pointers to GPU based solutions for the application.
- Study of OpenGLSL was started but not finished.

3.2.4 STM32 - IDE installation

- IAR Embedded Workbench ARM v7.10 was chosen and installed.

3.2.5 STM32 - SPI and DMA

- Successful communication test in slave mode (using DMA) with the Raspberry Pi.

3.2.6 Raspberry Pi - SPI

- Low level library installed. This allows using SPI directly commanding the processor (skipping the Linux kernel).
- Successful communication test in master mode with the STM32 as slave.

3.2.7 Raspberry Pi - Performance improvements

- A prioritized list (for the application at hand) of potential optimizations to have in mind:
  - Using the GPU for image processing.
  - Overclocking.
  - Shutdown unused modules.
  - Running the raspberry "bare bones", without Linux.

3.3 Pending

- **Manufacture of the camera mount mechanism:** The mechanism was not manufactured but it was decided that a simpler one will be used to see the effect of vibrations on line detection in general before building the final one.
- **DCMI test:** I spent some time turning the LCD off (because it conflicts with the DCMI pins) and deciding which of the board pins are the best to use. This had a software-side learning curve.
- **Region of interest:** The time that was planned to be used on implementing ROI to the vision algorithms was spent studying OpenGLSL because the implementation of ROI will be different on the GPU.

DCMI	Pin	Board	Pin	Board	Pin	Board
D0	PA9	Free	PC6	LCD-TFT & LCD-RGB		
D1	PA10	Free	PC7	LCD-TFT & LCD-RGB		
D2	PC8	Free	PG10	LCD-TFT & LCD-RGB	PE0	SDRAM
D3	PG11	LCD-TFT & LCD-RGB	PC9	ACP/RF & Touch panel	PE1	SDRAM
D4	PE4	Free	PC11	Free		
D5	PD3	LCD-TFT & LCD-RGB	PB6	SDRAM		
D6	PE5	Free				
D7	PE6	Free	PB9	LCD-TFT & LCD-RGB		
HSYNC	PA4	LCD-TFT & LCD-RGB				
VSYNC	PB7	Free	PG9	Free		
PXCLK	PA6	LCD-TFT & LCD-RGB				

Table 4.1: Pin options for each DCMI signal according to the microcontroller's datasheet and the peripheral they are connected to at the discovery board.

Conflicted pin	LCD TFT	LCD RGB	I/O	Action
PA4	VSYNC	VSYNC	I	Nothing
PA6	DB6	G2	I/O	Force input or float
PD3	DB11	G7	I/O	Force input or float
PG11	DB11	B3	I/O	Force input or float

Table 4.2: Discovery board DCMI non-free pins and the action to trigger at the target device to avoid problems.

4 MEMORANDUM 04

May 19 - May 23

4.1 Activities

4.1.1 Pin selection

Signal	SPI1		SPI4	
	Posible Pin	Datasheet	Posible Pin	Datasheet
SCK	PA5	Free	PE12	SDRAM
	PB3	System	PE2	Free
MOSI	PB5	SDRAM	PE6	Free
	PA7	ACP/RF	PE14	SDRAM
MISO	PB4	Free	PE5	Free
	PA6	LCD-TFT & LCD-RGB	PE13	SDRAM
NSS	PA15	Touch panel	PE11	SDRAM
	PA4	LCD-TFT & LCD-RGB	PE4	Free

Table 4.3: Options for each SPI signal (SPI1 and SPI2) according to microcontroller's datasheet and the peripheral they are connected to on the discovery board.

Signal	SPI5		SPI6	
	Posible Pin	Datasheet	Posible Pin	Datasheet
SCK	PF7	LCD-TFT & LCD-SPI & L3GD20	PG13	LED
MOSI	PF9	LCD-TFT & LCD-SPI & L3GD20	PG14	LED
	PF11	SDRAM		
MISO	PF8	L3GD20	PG12	LCD-TFT & LCD-RGB
NSS	PF6	Free	PG8	SDRAM

Table 4.4: Options for each SPI signal (SPI5 and SPI6) according to microcontroller's datasheet and the peripheral they are connected to on the discovery board.

Conflicted pin	LCD TFT	LCD RGB	L3GD20
PF7	DCX	SCL	SCK
PF8	-	-	MISO
PF9	SDA	SDI/SDO	MOSI

Table 4.5: Discovery board SPI5 non-free pins and the signal they carry. It is only necessary to keep the SS pins high for these devices.

Pin	Peripheral	Signal
PA04	DCMI	HSYNC
PA06		PXCLK
PA09		D0
PA10		D1
PB07		VSYNC
PC08		D2
PD03		D5
PE04		D4
PE05		D6
PE06		D7
PG11		D3
PC01	GPIO	CS Gyro
PC02		CS LCD
PF10		EN LCD
PC09	MCO	MCO2
PF06	SPI5	NSS
PF07		SCK
PF08		MISO
PF09		MOSI
PA05	TIM2	TIM2_CH1

Table 4.6: Final selections of pins to be used by the application.

5 APPENDIX

5.1 Gantt A

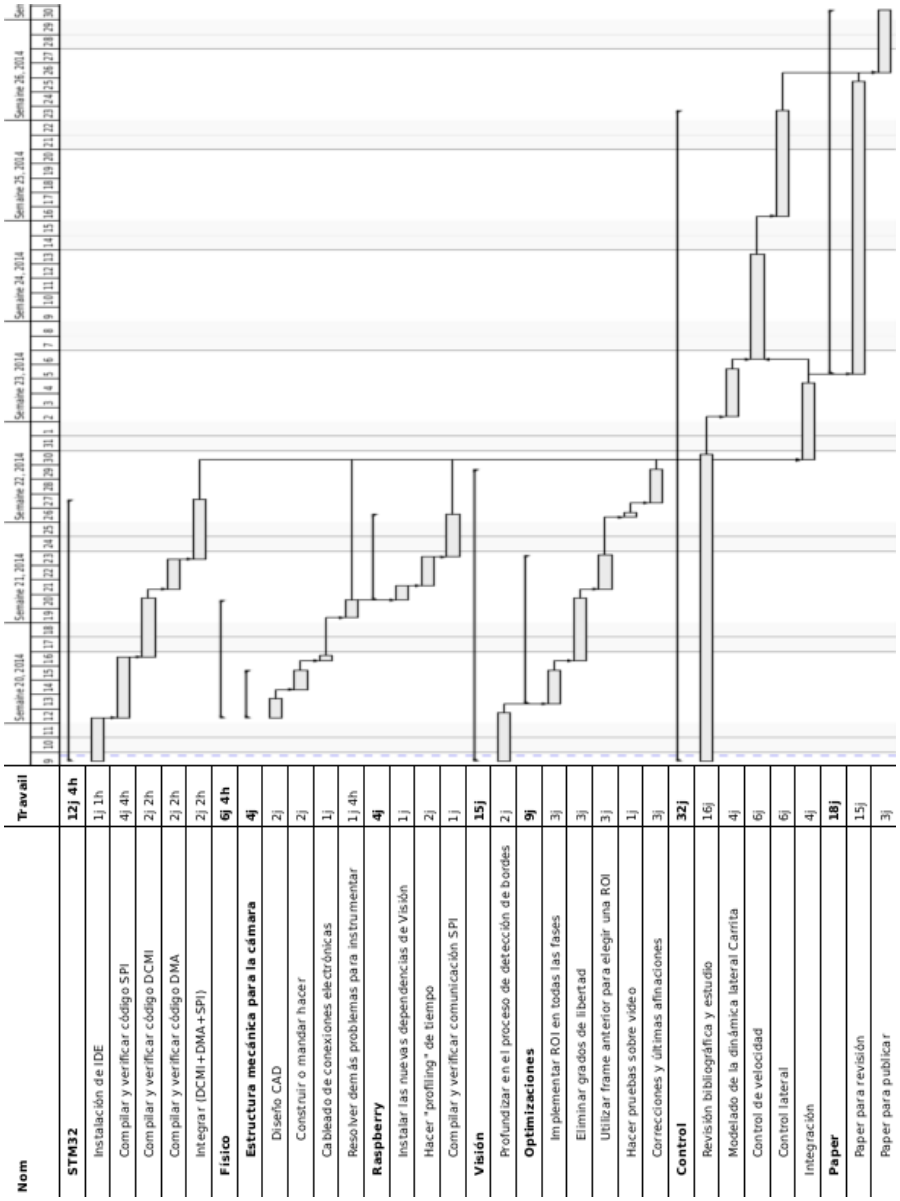

Figure 5.1: Gantt para los primeros dos meses de trabajo.

5.2 Camera mount mechanism drawings

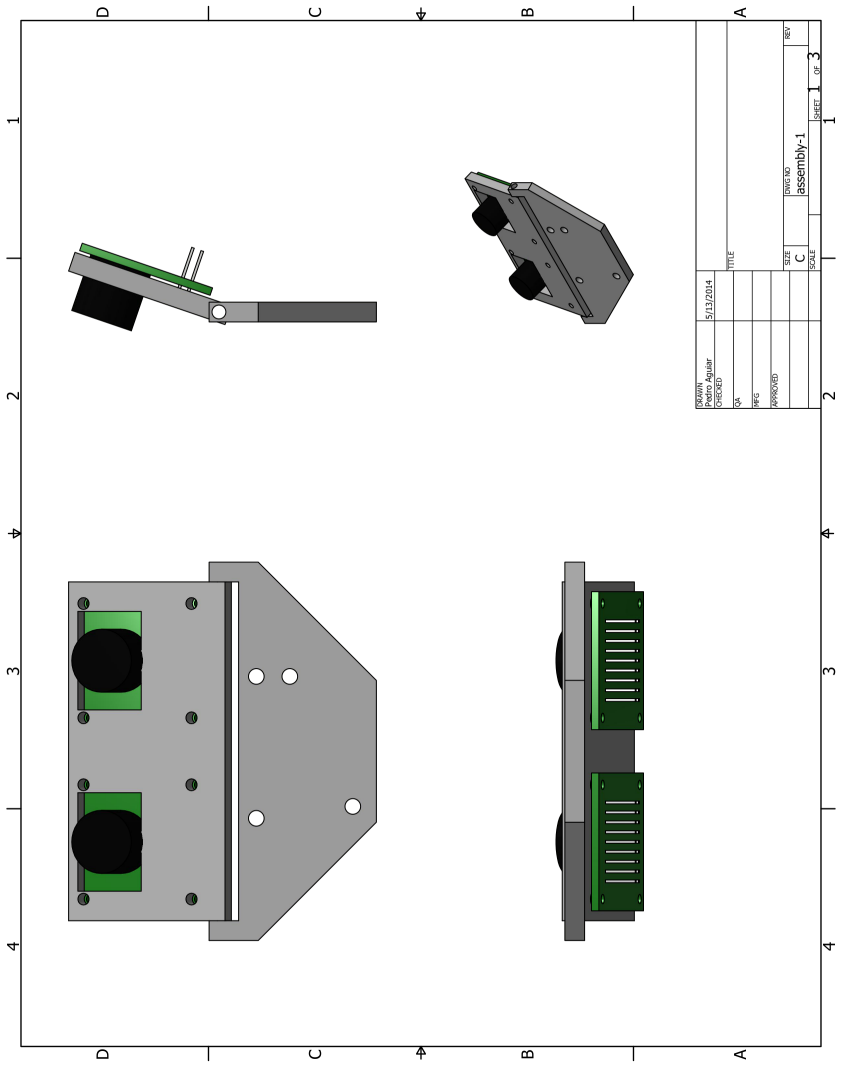

Figure 5.2: Camera mount mechanism drawings: Assembled camera mount.

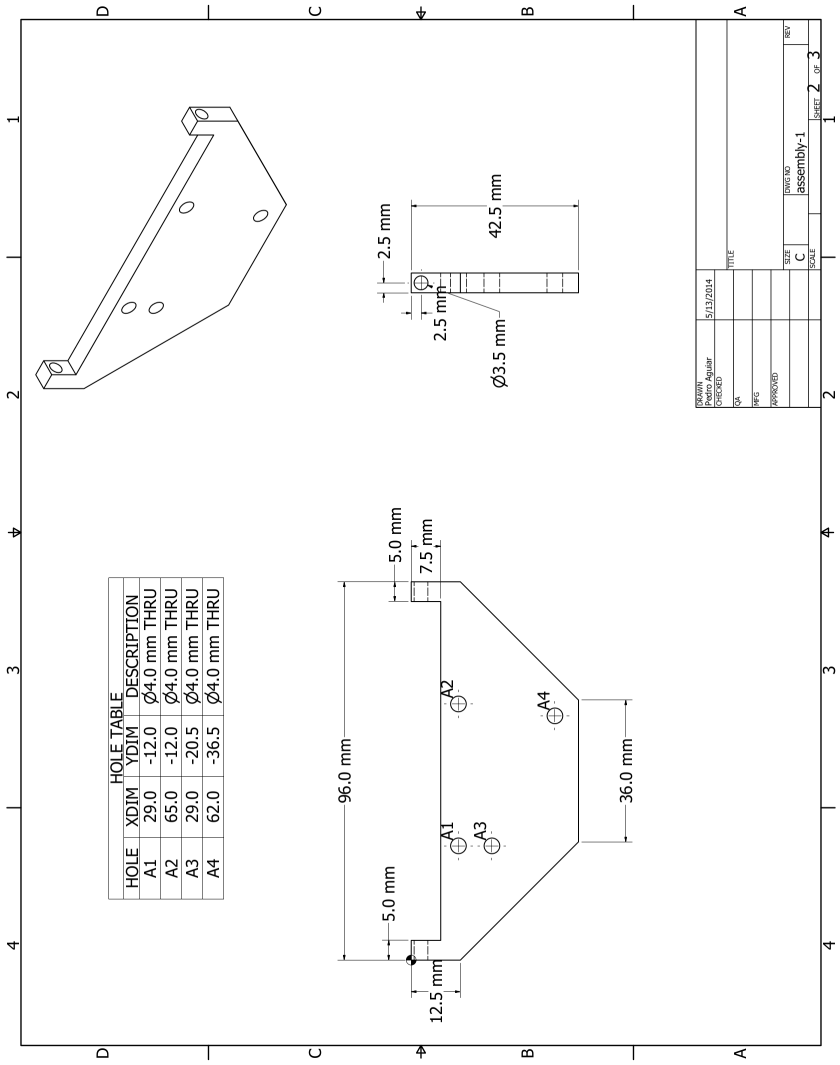

Figure 5.3: Camera mount mechanism drawings: camera part.

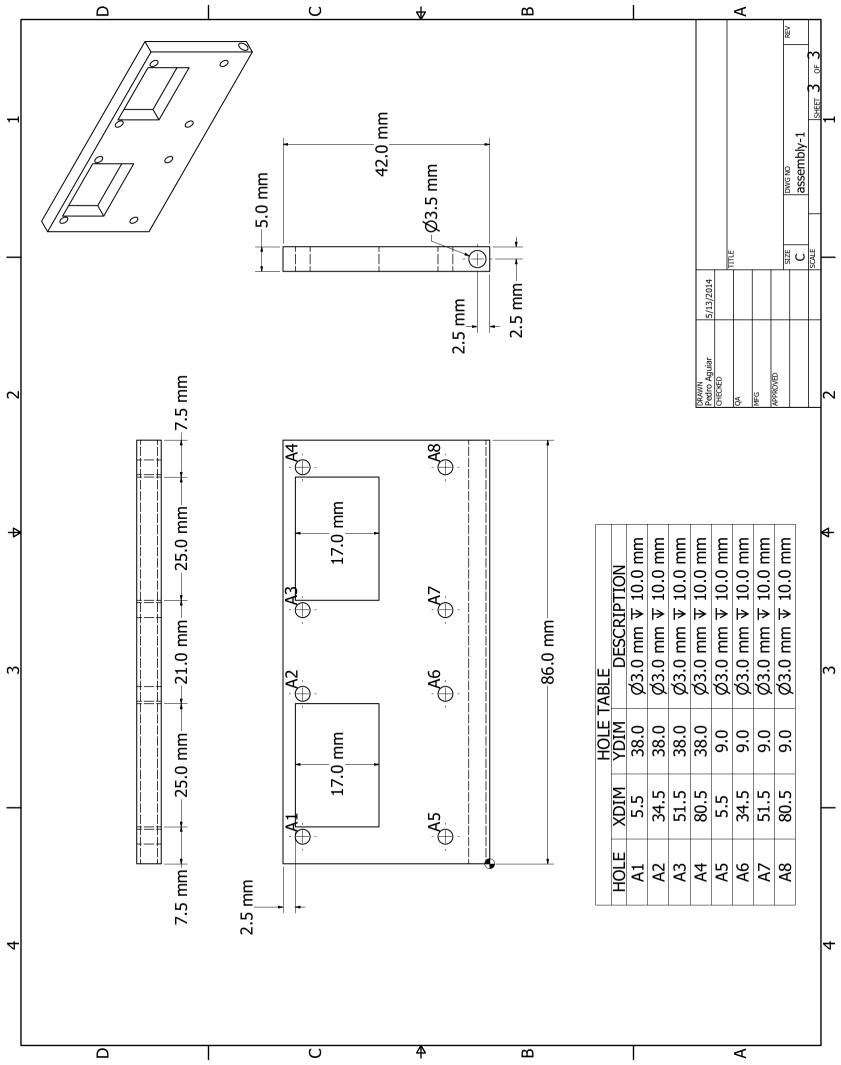

Figure 5.4: Camera mount mechanism drawings: Carrita's part.

References

[1] Chris Cummings, *Gpu accelerated image processing on raspberry pi.* http://youtu.be/yQZISXIFjaQ

[2] Markéta Dubská, Adam Herout, *Real-Time Detection of Lines using Parallel Coordinates and OpenGL.*