AUTONOMOUS NAVIGATION WITH DISCRETE MARKERS

Final report for Team 4

12/2/16

Contents

I) Overview – Derrian Glynn

A) Overview

This goal of this project was to develop a discrete navigation autonomous vehicle. The project was to move along a designated rectangular path with set markers to relay to the vehicle where to go at 1 mile per hour. The vehicle was to also have audio and visual output to relay to external parties about its movement.

B) Introduction

The Discrete Navigation Robot is built to sense ARUCO markers to follow along a designated path. The markers are detected through the camera attached to the system to follow the directions of the specific scanned marker. As the car travels it holds several devices to give information to outside parties about its behavior such as when it turns on the LCD screen. The car is controlled through a PWM servo system to limit its speed to 1 mile per hour. The car is powered by 2 power sources. One battery pack to power the Raspberry Pi and the screen and another battery to power the motor and the servo system.

II) Digital Systems – Rahul Chidurala

A) Main Control Unit

[6]

The Raspberry Pi 3 was chosen as the main microcontroller for this project.

Image processing can be very difficult to implement from the ground up, and always requires a lot of processing power and RAM to achieve results in real time.

This immediately eliminated many of the common microcontrollers that are used for similar robotics applications. The Arduino Uno is one of the most commonly used and supported microcontroller, but its specifications are not enough to perform real time image processing. The Arduino Uno uses an ATmega328P processor which has only 2 kB of RAM with a 16 MHz clock speed, and has storage of 32 kB **[1]**. An average image file with JPEG compression contains 500 kB, so the Arduino Uno is not able to store even the images **[2]**. The Arduino Mega 2560 board has 256 kB of memory, which makes it also insufficient for storing an image. The Arduinos are good for lower level programs without intensive processing, but are inadequate for image processing. The higher end Arduinos, such as the Galileo cost over a $100 and would take up the majority of the project's budget.

The Raspberry Pi 3 offers a lot more than the Arduinos and at a reasonable price. At $35, the Pi has a Broadcom BCM2837 System on Chip (SoC) with a quad core ARM Cortex-A53 at 1.2 GHz, and a Broadcom Videocore IV GPU, and 1GB of RAM. A microSD card is used for its storage and a typical microSD card of 8 GB

is more than enough to store the necessary images in storage and also process in real time using the 1 GB of RAM **[4]**. The Pi also has an HDMI output and auxiliary audio output, which made it ideal for setting up audio-visual outputs for enhancing the project. The Pi also has 40 General Purpose Input-Output (GPIO) pins available for signals ranging from simple pull up or pull down pins to complex serial communication such as I2C.

What makes the Pi even more appropriate for this project is that it contains a Linux based operating system on board. The Linux OS also allows support for multiple language support such as Python (the supported language for the PWM driver and GPIO pins) and C++ (for image processing). With a Linux OS, open source libraries can be used to perform image processing. OpenCV (Open Computer Vision) is a very popular, open source image processing library with plenty of support. OpenCV will allow the project to implement proven, well known algorithms to process images and extract necessary data.

B) PWM Driver

[6]

Initially, there was a simple microprocessor on the car that signaled H-bridges to control the direction of the main drive motor at one speed (10 miles per hour), and the included steering motor was only capable of binary turning. The project's specifications required that the car maintain speeds around 1 mile per hour while driving around the course, and a positional steering servo was necessary to make more controlled turns. The initial arrangement of the car did not allow for any easy customization for

controlling the motor and its speed. To control the speed of the drive motor, the Pulse Width Modulation (PWM) method was elected.

The Pi only has software PWM outputs which are unsteady, noisy signals and can cause jitter within the motors, so a PWM driver was needed to generate stable, clean PWM signals. Adafruit's PCA9685 is commonly used for generating PWM signals for the Pi. The PCA9685 is able to output 16 separate PWM signals each with a 12 bit resolution. The 16 channels are more than adequate for the two PWM signals needed: one for the drive motor, and another for the steering servo. The 12 bit resolution, which gives PWM resolution control of up to 1/4095, was necessary for precise speed control of the drive motor and for precise steering using the positional servo.

C) Motor Driver

The motor driver can handle up to 13 amps and this high current is needed for overcoming the beginning stall of the drive motor which is around 6 amps because the car is in a 4 wheel drive configuration. The motor driver has a PWM input for easy control of the motor speed, and a pin for reversing the direction of the motor. This motor driver was ideal for this car's motor and configuration.

III) Hardware- Brian French and Rahul Chidurala

A) The Vehicle – Brian French

The original car that was provided had many issues with it. The first issue that was immediately noticeable was the turning of the axles were bent. That became the first issue to overcome. The cheap plastic material of the car was another issue that was soon going to be an issue. Making any adjustments for later would be extremely precise work. Since the plastic was so thin and cheap any heat applied to the car would warp wherever applied. The battery life was also a huge disappointment. The 800mAh battery was going to provide a maximum of ten-minute lifespan, this

was immediately a problem being that the project specified that the end result must be able to perform its tasks for a minimum of two hours. The servo that was provided with the car was a binary turning servo which for the task at hand would end up being a very large issue. We need softer turns than what a binary servo could provide. These were the first problems visually and immediately known.

B) Servo – Brian French

[6]

As was mentioned in the section discussing the car, it was very evident that the servo provided was not going to be able to help in the final product. The servo provided while yes it was binary turning, it was also extremely small and had nearly no power to turn the car any amounts. It was actually unknown how it was managing it turn the car beforehand given that it had virtually no control.

This was replaced with a much larger 1/5 servo from the 1/8 servo that was provided. This servo was also able to outperform in nearly every way comparatively with the original servo. This servo is a positional servo, which accomplishes the tasks at hand. Giving the ability to partially turn according to what was inputted allowed for a better and smoother turning when it came to the corners of the project.

IV) Car Modifications –Brian French

The main tool used for the project was the Dremel 2001. This device has 41 different attachments used on this project. This varies from a 1/8” Drill bit to a 3/4” flat head cutting piece. Throughout the project it was very evident that car was going to needed to be modified very severely. There was hardly any room in the car for any extra devices components or any larger devices. (I.E the servo). The modifications of drilling, cutting, smoothing, and many other modifications took over 140+ hours of work. There were many original concepts and ideas of where thing would be placed but as time progressed it was very apparent that the original placement of parts was not going to work. Tangling of wires as well as fitness of components in such a small compacted area. There was not much room for error especially with the warping of the plastic. There were even areas that needed to be cut out to fit parts and a hinge system added to allow the easy access to the voltage regulators, and other components

V) Sensors – Derrian Glynn

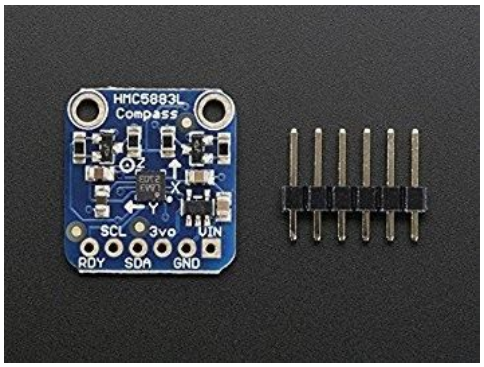
A) Camera

[6]

For being able to recognize the discrete markers, the vehicle would need a camera to recognize them. Since OpenCV is set to be able to process live images from a USB port, a USB camera was selected to to be used to be the visual feed for the image processing. The Logitech HD Webcam C525 was selected for the project as it had the best resolution and widest angle for its price range.

The camera was connected to the PI through USB, mounted on to the car on top of the hood of the car and turned to the left. The camera had a capture frame rate at 7 per second and was used to recognize the markers. As the markers were on a slant, to the left of the car's track, the angle would allow the camera to get a better view of the markers on the ground.

B) Compass



[6]

To help keep the compass moving in a straight direction along the markers, the vehicle needed a secondary sensor to help keep the car going in the intended direction. Several sensors were considered for the project such a gyroscope or a GPS. However, a compass was selected for its accuracy, functionality with the Raspberry Pi serial pins and price.

The Adafruit Triple-axis Magnetometer Compass was selected for the navigation sensor. By communicating with it through I2C, the compass relays its 'X', 'Y', and 'Z' coordinates. The coordinates can be used to calculate the coordinates of the direction the compass is pointing in. Potentially the car could have used the compass in a PID controlled stirring system to navigate itself to drive straight. However, due to the magnetic field of the equipment on the vehicle, the compass was unable to give out correct information.

VI) Device Output – Derrian Glynn

A) Speaker

[6]

For the vehicle to relay audio information to the user, the system would need a speaker in order to output any audio. The PI could output audio simply through the 3.5mm audio output on the board. However, the speaker would need to either be powered off of the Pi itself, or have its own power supply. The speaker would also need to be loud enough to broadcast the audio from several feet away and be compact enough to fit on the car itself.

The Otimo Mini Rechargeable Bomb Speaker was selected as the audio speaker for the vehicle. The speaker has its own internal battery that can be recharged by USB, which could be done either by connecting it to the PI itself, or from one of the batteries. The speaker was spherical in shape with a diameter of 2.2 in, allowing it to fit inside of the car.

When commanded, speaker would play audio

B) Screen

 [6]

As the vehicle needed a visual output to relay information to the user, a screen was determined to be the best selection for the project as it would be the most efficient way to give information. The Raspberry Pi had several ways for a screen to connect a screen with either through HDMI, or the serial pins. The HDMI port was the selected port as the port was the easiest to setup and had the most options. The screen used for the project was the **HDMI 7" 800x480 Display Backpack** as it would the biggest screen to work for the project without being a massive power drain or too expensive.

The screen was used, not only for debugging the PI, but also to give the user information as the car runs, such as the direction the car was going, as well as the visual information. If the car detected a forward marker, the screen would be able to relay an arrow pointing forward. The screen would also be used to output the images of the locations that were associated to the stop.

VII) Power System - Brian French

A) Power Source

[6]

From the section discussing the original car parts there was a brief discussion over the very disappointing battery. The 800mAh battery was going to provide a maximum of ten-minute lifespan was one of the first things to go. To have a better lifespan there was a lot of research that went over extended the lifespan as well as finding a more efficient battery. Towards the middle of the year the battery that was changed to was the 2,000mAh NiCd. (Nickel Cadmium) This allowed a 5 hour lifespan and allowed to have nearly zero loss in energy dissipation.

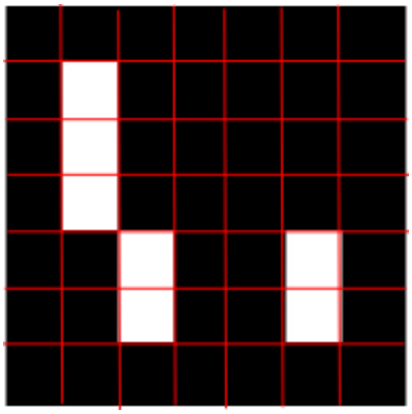
There was also a second battery that was implemented to supply a separate power source to give the raspberry pi the amount of energy needed to run. That was simply bought according to the specs that the raspberry pi needed to have. (KMASHI 10,000 mAh external power pack)

B) Voltage Regulators

[6]

As with every project there is a need to provide power to each component, however all components have different power specs they need to follow. There were two voltage regulators that were built to supply 12 volts to 12 volts, (The reasoning for this was to make sure there were no spikes in voltage to destroy any of the mechanisms) as well as a step down voltage regulator from 12 volts to 5 volts. The servo needed 5 volts to give it the amount of energy it needed to turn the car, as well as the motor driver needed 12 volts.

VIII) ARUCO Markers – Rahul Chidurala



ArUco marker example. The red lines show the bit configuration.

ArUco is a library that relies on fiducial markers. Fiducial markers are used as a point of reference for image processing applications. ArUco is minimal and relies on OpenCV, which makes it compatible with essentially any system that can utilize OpenCV. The markers can have up to 1024 unique IDs because each marker contains 5 words of 5 bits each (3 bits of the 5 are used for error detection).

After performing the image processing and successfully identifying a marker, the ArUco library provides the marker ID, and the rotation and translation vectors that describes the relation of the center of the marker to the camera location. The rotation vectors seemed promising to align the car to the track. The translation vector's z coordinate corresponds to the distance of the marker from the camera. These markers and the information about their pose (rotation) and translation vectors can be used to create the behavior of the autonomous car.

IX) Program - Rahul Chidurala

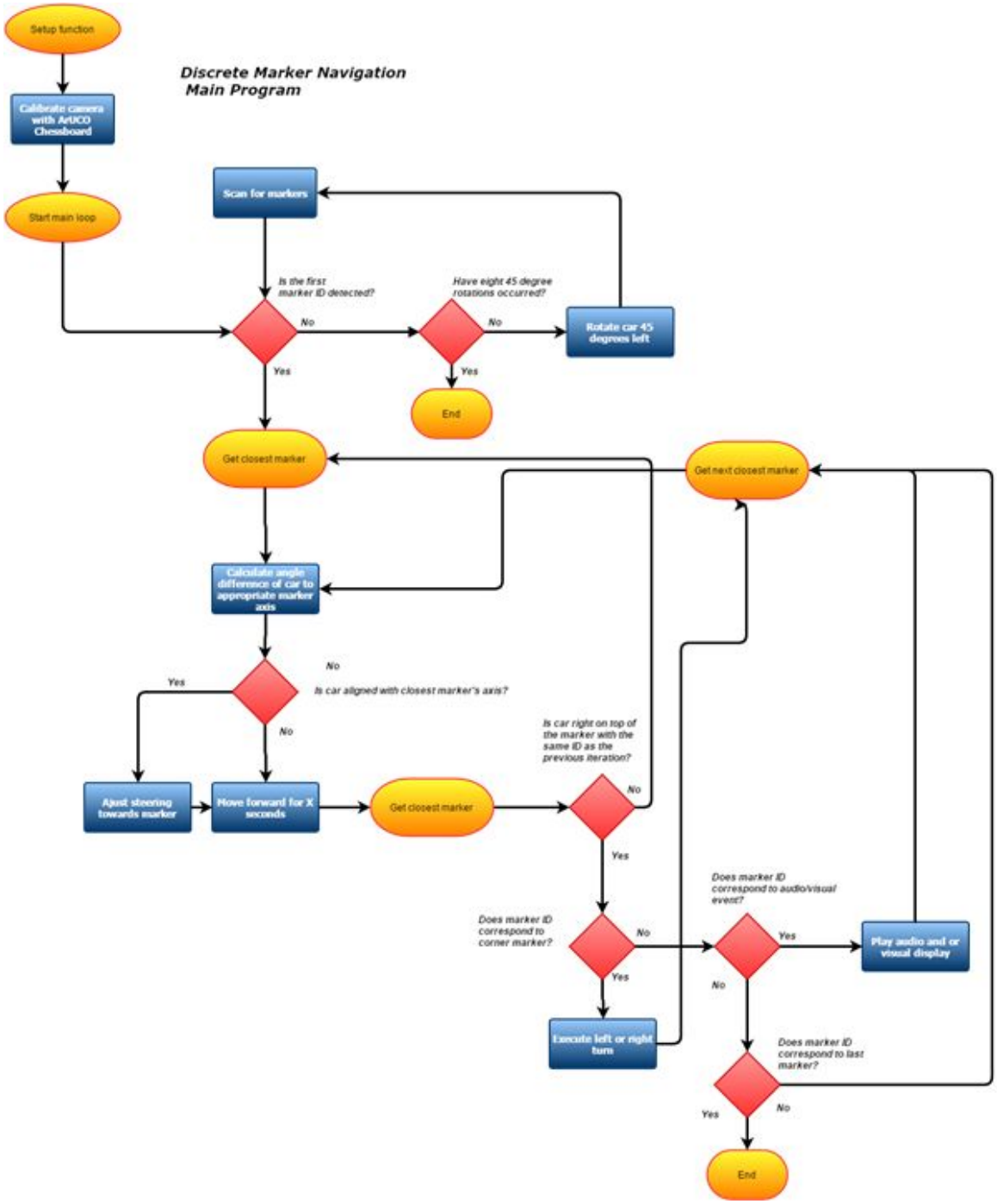
The main program is scripted in Python. The PCA9685 PWM driver, which controlled the drive motor and steering servo, has an I2C interface. The Pi's native Python libraries support I2C and Adafruit supported the board with Python libraries that allowed for an easier use of the PCA9685 PWM driver board.

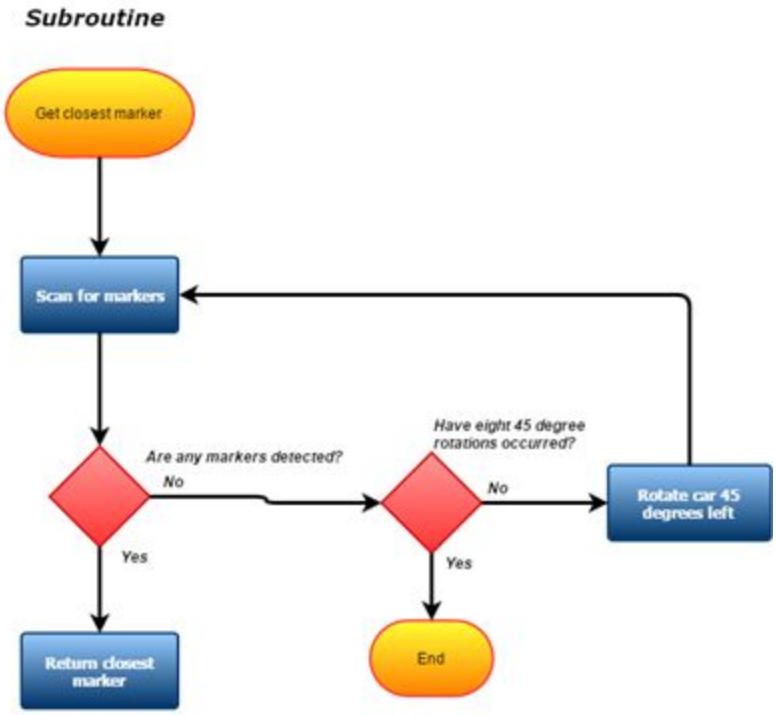
The image processing was written using OpenCV and the ArUco libraries which were in C++. Code Blocks was used to build the ArUco project to be built on the Pi, and a new module was created for this project to process the markers.

Piping was used to connect the image processing aspect of the project with the hardware and motor controls. The POpen() function allows Python to open a new instance of this project's custom ArUco marker detection program, and the standard output of the custom marker detection program was able to be read by Python through a blocking buffer. The custom marker detection program was further modified to export all the detected markers in JSON. The JSON can then be easily parsed in Python using the native JSON libraries.

Threading was essential for this program structure to work because reading the pipe from the custom marker detection program was a blocking action, and the car needs control over its motors at all times. Issues can arise without threading when the main program signals the car to continue forwards and right after waits to read JSON from the pipe, then the car will be stuck in its current state of going forward until a new marker is detected and JSON is pushed to the pipe to the main program.

Main Program Flowchart





X) Integration – Derrian Glynn

With all of the parts necessary, each of them had to be combined in a specific manner in order to function together for the specifications of the car. In order for the car to be Starting with the base, the NiCd battery was placed in and connected to the 2 voltage regulators to give a 6 V and 12 V source for the rest of the devices. The 12V source was used to power the Motor driver of the car through its input port. The motor driver was then connected to the motor itself to relay from the main control unit when to turn on or off. The 6 V rail was connected to the servo which was attached to the axle of the front wheels to turn them as directed. All of these are mounted within the cavity of the vehicle for the most direct access to each other.

Both the motor driver and the Servo were connected to pwm ports on the PWM driver. The PWM Driver is mounted on the side of the car and is also connected to the 6V for its voltage source. It

relays the commands from the main control unit to the motor controller and the servo. The PWM driver also has an I2C communication port to let another device communicate through it, such as the potential compass or other devices to the main control unit.

The PWM driver is connected to and controlled by the Raspberry Pi. The Pi is the central control hub and manages all of the other devices. The Pi can be controlled through like a regular computer by connecting an USB keyboard and mouse as well as any HDMI screen or audio device with a 3.5mm Auxiliary connection. For the project however, the screen and speaker can be used, especially on the car. As well as being able to network into the PI by either VNC or SSH methods. The Pi and the screen are both powered by the KMASHI battery through their micro USB ports to the battery's USB ports. The camera is connected to the PI through USB.

While the Pi was mounted on top of the frame of the car and the KMASHI battery mounted on the back, the other devices, such as the screen, the camera and the speaker are mounted on top of the plastic cover of the car. The camera is mounted onto the front of the car, through a small opening on the front of the plastic. The Screen was mounted onto the back of the plastic with screws, facing up. The Speaker was mounted on the top of the car by hanging the ring from the top of the plastic from the inside.

XI) Improvements – Brian French

While the vehicle performed relatively well, there are a few places that could be improved upon for the vehicle for it to be at the best functionality.

Better Frame - The better frame would have helped in several areas. The space needed to fit components, the axles wouldn't have been bent and could avoid unnecessary turning. The original car that was given had a slightly bent axle and caused a lot of issues. With a new frame the results of the runs would have been more consistent and would have yielded better results.

Wide angle Camera - The camera originally was a good fit for the project at first, however after some time it was realized that the angle of the lens was not wide enough. With a larger angle of viewing it would have been able to view the markers even when the car would drive slightly off course. Now it could be argued that with a better frame it would not have been necessary to have the wide angle lens but with no budget it would have proven to be beneficial to have both.

Zero Degree turning - This would have helped in several areas as well. With the original car the turning was an angle from the front axle with the rear propelling the car forwards allowing the car to turn, with limitations of course. Zero degree turning would allow the car's front axle wheel to turn in other directions allowing a seamless turn. Within the program there would have been a lot of uncertainty cleared up with the turning. An issue that was run into much later in the year was the turning was inconsistent, again this could have been solved several ways but dude to the time and money constraints it was not a reality.

Gyroscope - The compass ended up being too hard to implement with this particular project due to magnetic interference from the servo. A gyroscope would have been the solution early on to avoid that problem while still given us the orientation and directional readings that were needed.

Bigger battery - Who doesn’t always want a bigger battery? A bigger battery would have meant a longer active time for the car itself. There wasn't a very large issue with this, the car was still able to run for hours on end but the issue still remained that after many hours of demonstration the car ran down and was unable to perform at its full potential whenever the time came.

XII) Conclusion- Derrian Glynn

After several months of construction, the vehicle was able to perform to best of its capabilities. The vehicle is able to turn drive itself with the use of all of the devices and output the correct information

as directed. However, the car had issues with turning properly due to lack of precision from sensors.

Still the vehicle was able to drive and react to the discrete markers and drive autonomously.

XIII) Bibliography

[1] Article title:Arduino - ArduinoBoardUno Website title:Arduino.ccURL:https://www.arduino.cc/en/Main/ArduinoBoardUno

[2] Article title:All About Digital Photos - Digital Photo File TypesWebsite

title:Rideau-info.comURL:http://www.rideau-info.com/photos/filetypes.html

[3] Article title:Arduino - ArduinoBoardMega2560 Website

title:Arduino.ccURL:https://www.arduino.cc/en/Main/ArduinoBoardMega2560

[4] Article title:Raspberry Pi 3 is out now! Specs, benchmarks & more - The MagPi MagazineWebsite title:The MagPi

MagazineURL:https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/

[5] https://www.uco.es/investiga/grupos/ava/node/26

1. Main ArUco paper:

"Automati generation and detection of highly reliable fiducial markers under occlusion"

http://www.sciencedirect.com/science/article/pii/S0031320314000235

2. Generation of marker dictionaries:

"Generation of fiducial marker dictionaries using mixed integer linear programming"

Article title: Generation of fiducial marker dictionaries using Mixed Integer Linear Programming Website

title:Sciencedirect.comURL:http://www.sciencedirect.com/science/article/pii/S0031320315003544

[6] Article title: Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & moreWebsite title:Amazon.comURL:https://www.amazon.com/