Supply and Demand Parking Meter - Design Review

Project #33

Adam Barbato - barbato2
Nick Johanson - njohans2
TA - Yuchen He

Table of Contents:

1. Introduction

1.1 Objective

Parking in dense urban areas is a problem where no obvious solution exists. Many approaches attempt to make it easier to find parking spots, however we believe the problem is that when people spend copious amounts of time searching for the spot it doesn't exist. Thus, the real issue is that there isn't enough parking, or perhaps parked vehicles are inadequately distributed. If parking is equivalently priced across an entire city people are incentivized purely on the distance between their parking spot and their destination, so they park as close as possible. This results in particularly high parking densities near popular locations and the inevitable 30 minute quest for a parking spot.

We propose a parking meter which can be implemented with a backend capable of monitoring prices across a city and creating economic incentives for a more favorable parking distribution. This parking meter would monitor the usage of spots it's responsible for and the prices that were applied, then this information would be relayed to another computer which will alter the prices proportionally. Prices would be reduced in areas with low density and increased in places with high density. The algorithm will have a goal density in mind and attempt to reduce the maximum density below that goal.

1.2 Background

Finding street parking is an annoying problem anybody who was driven into a large city has encountered. However, bigger problems with free, or underpriced, parking was laid out in the book *The High Cost of Free Parking*, in which UCLA Professor Donald Shoup explains that free parking encourages driving, which encourages that more free parking be built, which encourages more driving and so forth [2]. This eventually leads to city maps that are largely empty parking lots. So not only could our project, if implemented, help people plan their trips and daily routines better by offering them choices of parking spots and prices but it could also help cities restore walkability.

1.3 High-level Requirements

- Correctly identify the number of cars within the FOV of the camera(s) at 5 minute intervals with at least 90% accuracy in sunny weathered daylight conditions.
- Reduce the maximum density in a simulation to below 85% or the difference between the maximum and minimum density by 15%.
- Have Camera, CV, and communications running on custom built hardware.

2. Design

2.1 Block Diagram

Each parking meter will contain an ATMega2560, a camera, a voltage regulator, an Xbee transceiver, and the 16MHz oscillator. The ATMega2560 will store images and run the computer vision algorithm for recognizing cars. The camera will take images at set intervals and send them to the ATMega2560 for processing. A voltage regulator will ensure the external DC power is regulated to 5V. The Xbee transceiver will serve as the network link to the “server” computer. Lastly, the 16MHz oscillator will serve as the clock for the ATMega2560.

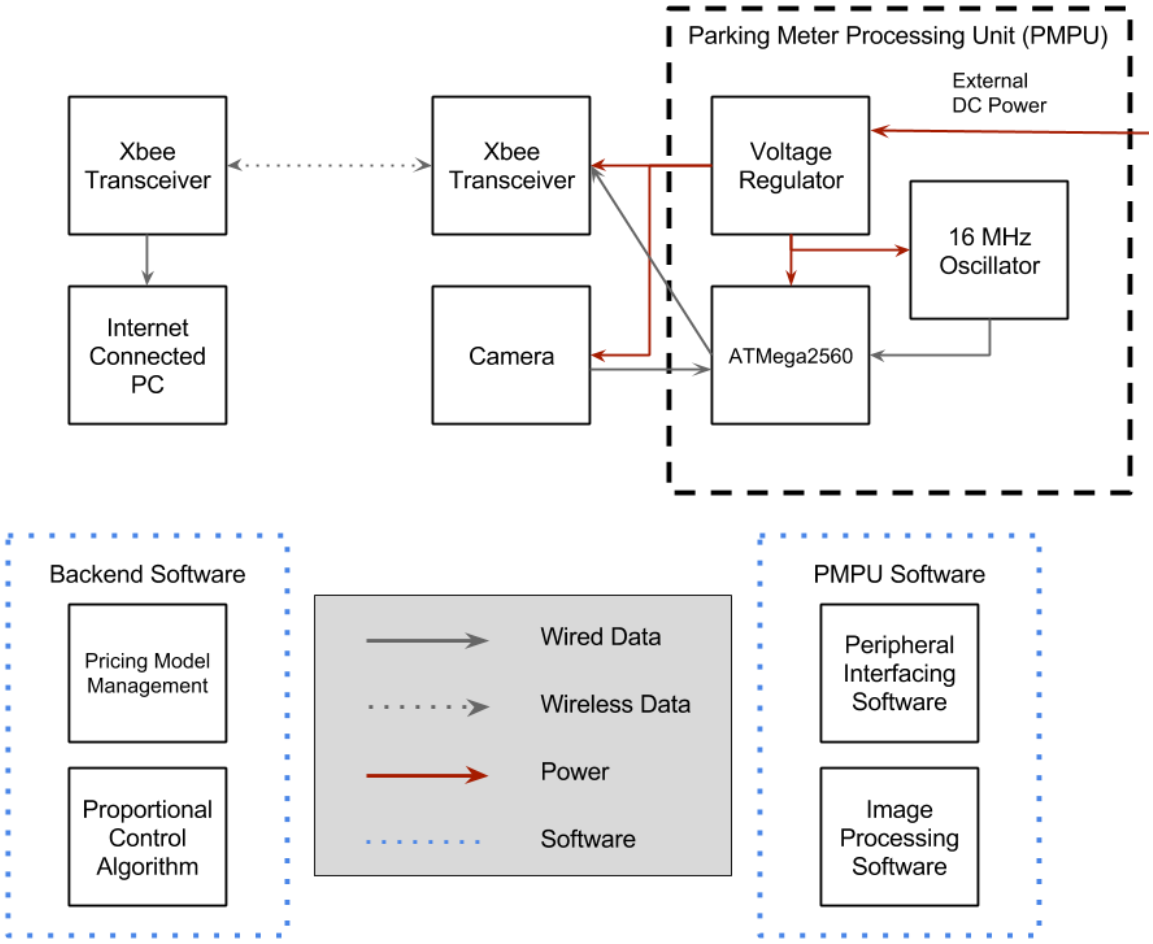


Figure 1: Block Diagram of project

2.2 Physical Design

The design for the physical parts of this project is fairly simple due to the low number of physical components. It will be entirely self-contained within a parking meter shaped box that will contain the camera module, PMPU and one of the two Xbee transceivers. This group of hardware will be placed around 20ft from the cars it's going to surveil, as explained in the calculation below, and will see each of its three parking spots from a different angle to ensure close proximity to the cars. The other transceiver will be located up to 100ft away from the parking meter and will be directly connected to an internet connected PC.

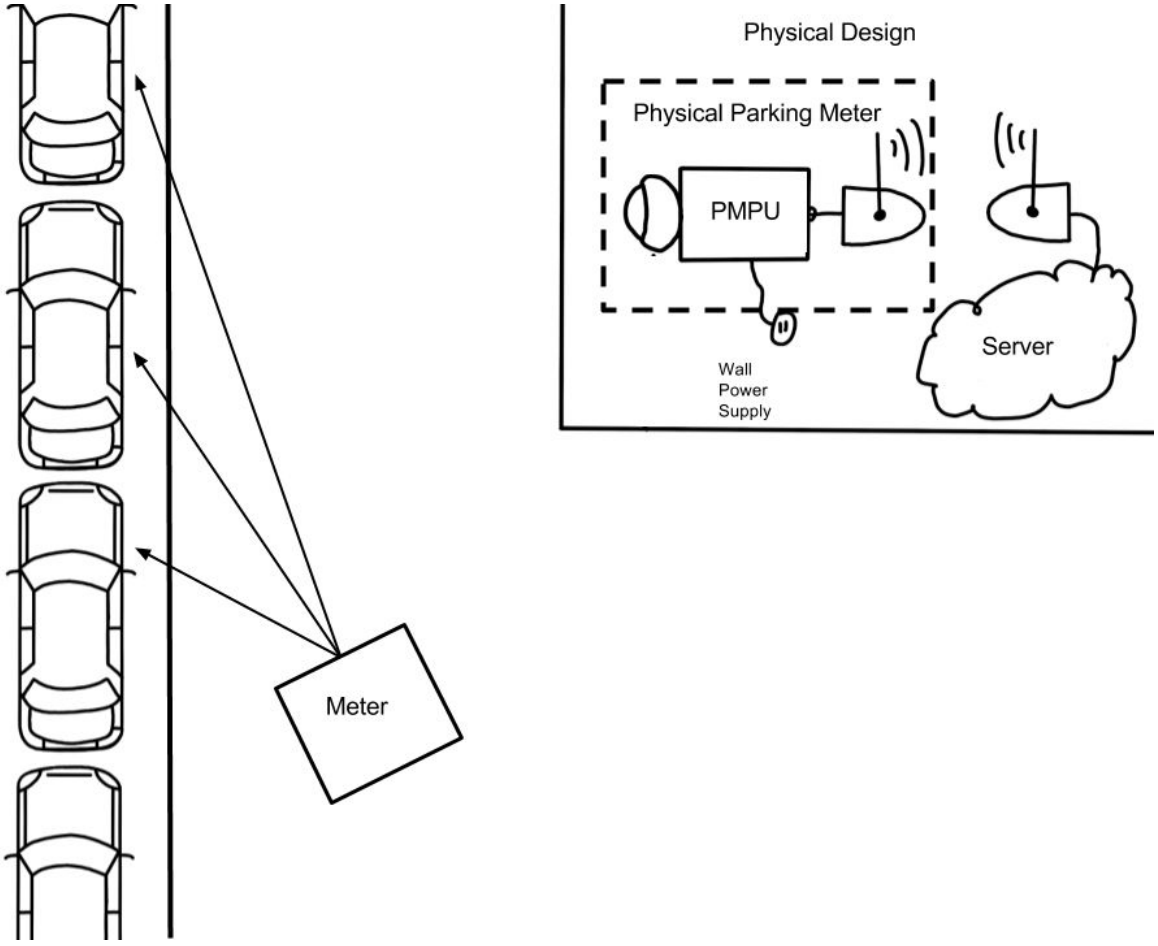


Figure 2: Physical Design of Parking Meter

2.3 Calculation for Placement of Parking Meter

Since this project is designing a parking meter, part of the physical design is where it will be placed on the street. The optimal placement is as close as possible to the street while still being able to see three cars, as specified in the requirements. We calculated that optimal placement as shown below in Figure 5. Where $L_{Parking}$ is the regulation size of a parking space which is 18ft [1], $2L_{Parking}$ is twice the regulation parking space size and the length the camera needs to see so that it sees at least half of every spot, θ is the angle of the camera lens: 60° or ⅔*π radians, and x is the optimal distance to the curb to place the meter. Using that information we can find x by the following equations:

$$Tan(\theta) = \frac{2L_{Parking}}{x}$$

$$x = \frac{2L_{Parking}}{Tan(\theta)} = \frac{36}{Tan(2\pi/3)} = 20.78\,ft$$

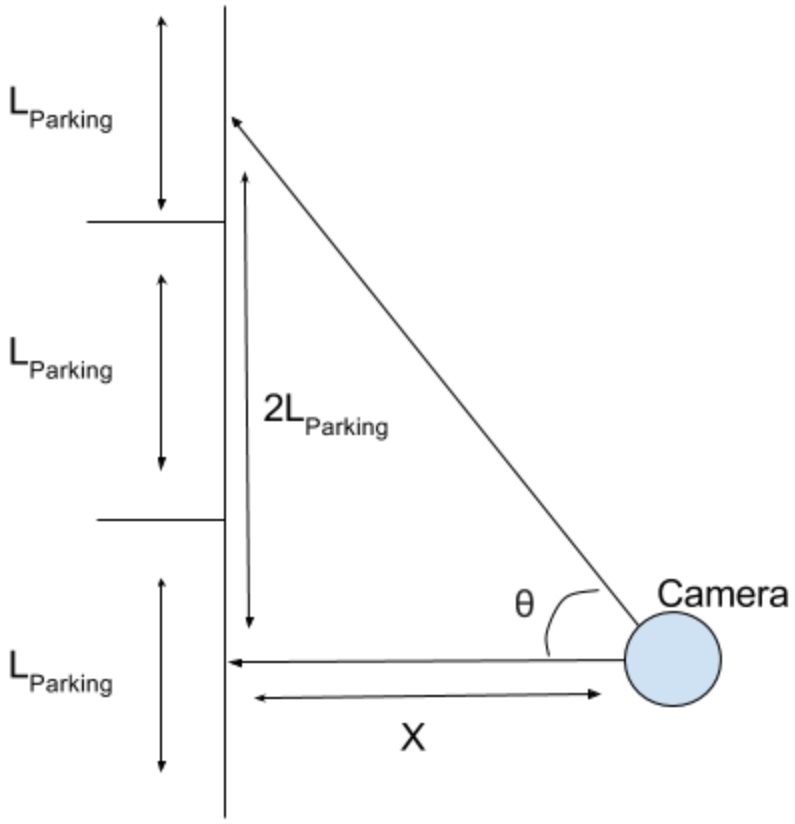


Figure 3: Reference for calculation of optimal meter placement

2.4 Camera (2 points)

- The camera is the piece of the device that collects all the data for the system. It will be mounted on the parking meter at a 60 degree angle from the street as to get the best angle for viewing multiple cars.
- At least every five minutes it will receive a signal from the PMPU unit to take a QQVGA (160 X 120 pixels) picture of the three parking spots within its view and send the picture back to the processing unit. This resolution was chosen so multiple pictures could fit in the PMPU's 256KB of storage. The timing was chosen because this camera model takes up to a minute to transfer a photo, so the extra timing was added for processing. The slow camera was chosen due to the constraints of the project needing only large time intervals.

Most of the camera's functions are requirements on the software that runs it, not the camera itself

Requirements	Verifications
1. Angle and placement of lens must allow three cars to be captured in one picture	1.1 As shown in Figure 5 below, make sure camera is positioned such that 60 degree angle of lens could capture three cars. 1.2 Hook up camera to external display via video-out pins and ensure that three cars are visible
2. Must run on 5±.5V <500mA	2.1 Hook up to 5V power regulator using 5V and GND pins. Ensure that the camera boots up via LED indicator.

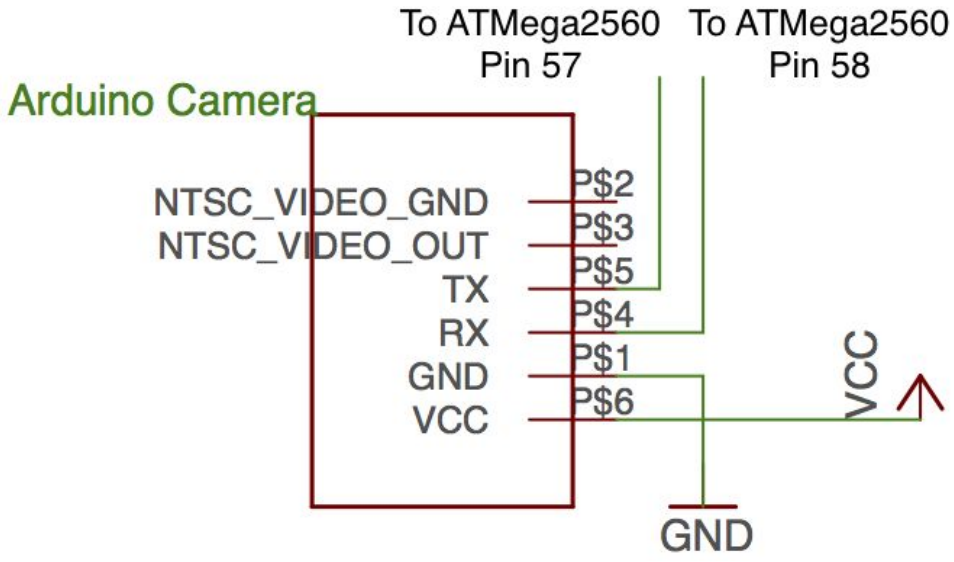


Figure 4: Camera Circuit Diagram

2.5 Xbee Transceivers (2 points)

- This unit actually consists of two physically separate radios, one connected to the PMPU and one connected to an internet connected computer that's away from the street.
- This unit is responsible for providing a wireless connection between the two parts it's connected and allows the PMPU to operate without an internet connection. This is to be implemented with an Xbee Series 1 radio solution that uses proprietary radio software on a 2.4 GHz channel has adequate bandwidth for this application.

Xbee Transceiver (2 points)
Most of the Xbee's functions are requirements on the software that runs it, not the Xbee itself

Requirements	Verifications
1. Must run on 5±.5V 2A	1.1 Hook up to 5V power regulator using 5V and GND pins. Ensure that the Xbee boots up via LED indicator.
2. Must be able to transmit data both directions between the parking meter and a waiting computer within 50 ft of the parking meter	2.1 Hook up one Xbee to Arduino or PMPU and the other to a laptop using the provided Xbee software 2.2 Use our software to transmit continuous data while moving Xbees away from each other 2.3 Confirm on the laptop that the Xbee receives data until at least 50ft away
3. Must transmit at least 1 kb/s	3.1 Repeat verification 2, ensuring that the transmission rate is greater than 1 kb/s using the Xbee software on the laptop

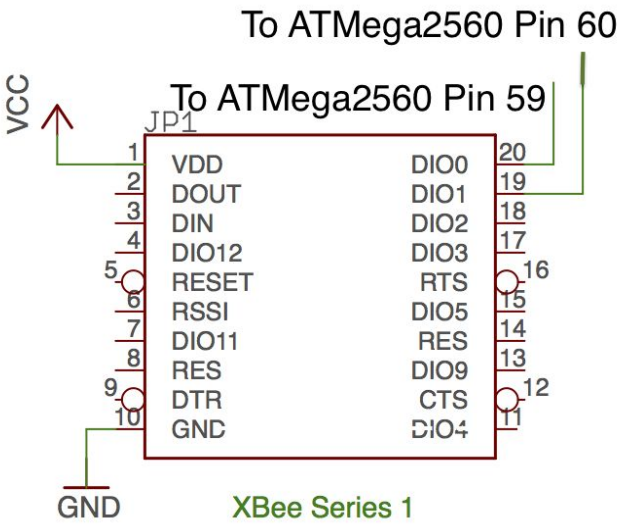


Figure 5: Xbee Circuit Diagram

2.6 Parking Meter Processing Unit (PMPU) Hardware (6 points)

- The PMPU is the main processing unit on the Parking Meter itself and consists of three major parts: ATMega2560, a 5V regulator, a 16MHz oscillator circuit. It also includes smaller circuits for resetting the device and an “ON” LED indicator. It provides power to both the camera and the Xbee Unit and controls both of their actions. A more detailed circuit schematic can be found in Figure 6 and 7
- The ATMega2560 was chosen as the main processor because it is the most powerful Atmel chip that still allows the Arduino bootloader to be used on it.

Requirements	Verifications
1. Must be able to regulate incoming 12V DC to at least 5±.5V and provide at least 3A output	1.1 Hook up 12V DC power supply 1.2 Measure output current and voltage, pins 80 and 99 on ATMega2560, ensuring that they are within the acceptable ranges
2. Must produce a 16±.5MHz clock signal	2.1 Hook up output of oscillator circuit, pins 33 and 34 on ATMega2560, to oscilloscope and measure the frequency for at least 30 seconds 2.2 Graph data and ensure it is all within the bounds
3. ATMega2560 must run on 5V <1A while doing all processes	3.1 While circuit is powered, ensure that “On” LED light is lit from the ATMega’s output

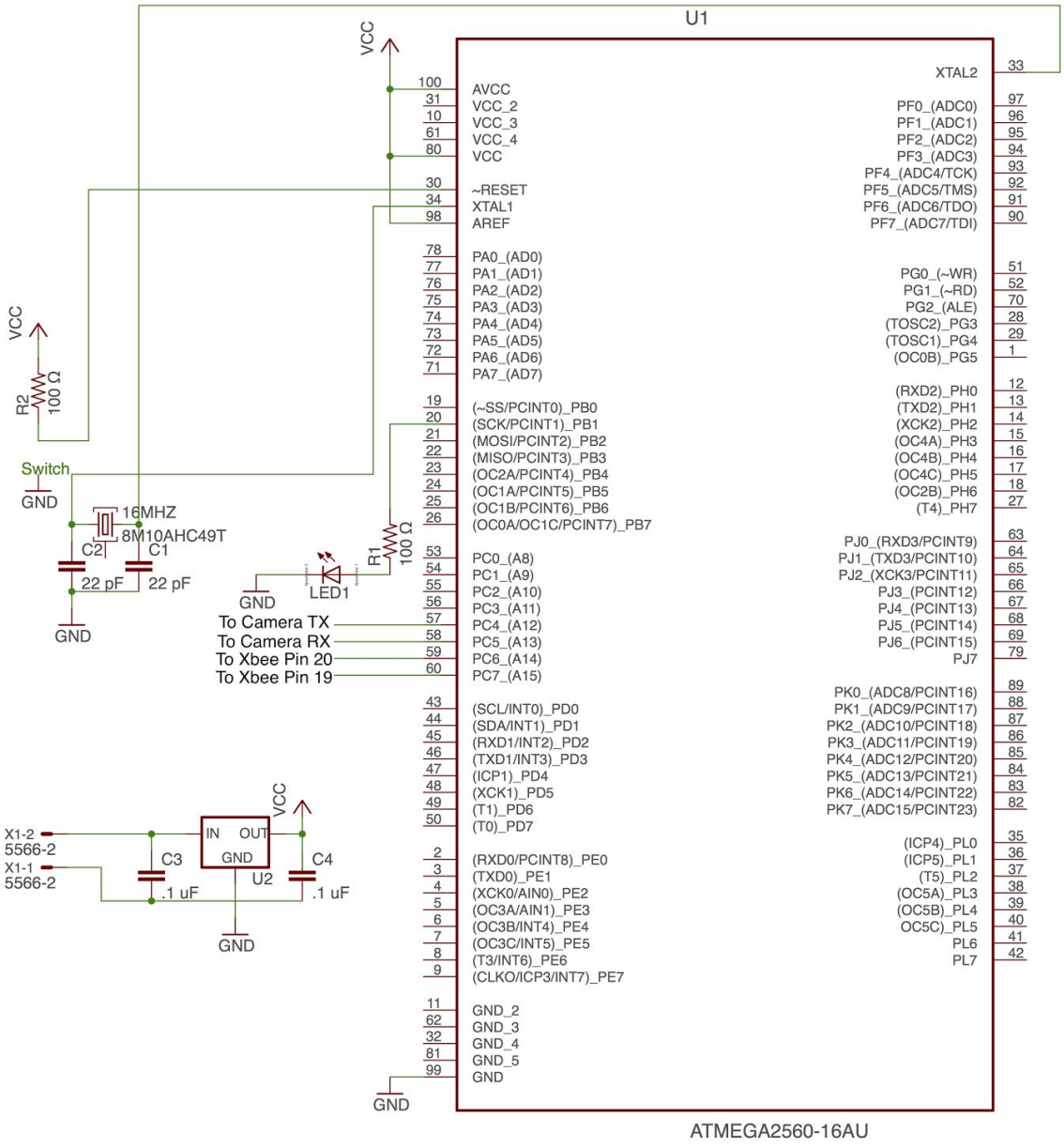


Figure 6: PMPU Circuit Diagram

2.7 Parking Meter Processing Unit Software

- This is the software that runs on the ATMega2560 in the PMPU hardware. This software is responsible for interfacing with the camera and the Xbee and does four main operations:
    - Instructs the camera to take a picture and saves the picture to memory
    - Using a pre-stored picture of the area with no cars, performs a Frame Differencing background subtraction of the new picture
    - Performs an FFT of the newly subtracted picture and compares it to a pre-stored spectrum to ascertain how many cars are parked in the spots it is monitoring
    - Send parking data and receive price information from backend software via Xbee radio

The PMPU Software is much more linear than the backend software, as is fitting for a microprocessor. The software mainly goes in a loop of: take picture, process picture, transmit data. The first thing the software does is request the camera to take a picture and transmit it to the ATMega. This can take up to a minute so the software will continue to check if the process is done. Once it has the new image it will load the old reference image and begin the background subtraction process by using a “Frame Differencing” algorithm. It then crops the newly subtracted image into three pre-defined sections that represent each parking spot. It then performs an FFT on each new image and compares it to a pre-stored spectrum to determine if a car is parked there. Once completed all images, it sends the data to the backend and waits for a response which may contain new price info. Once the response is received, it updates prices if necessary and begins again.

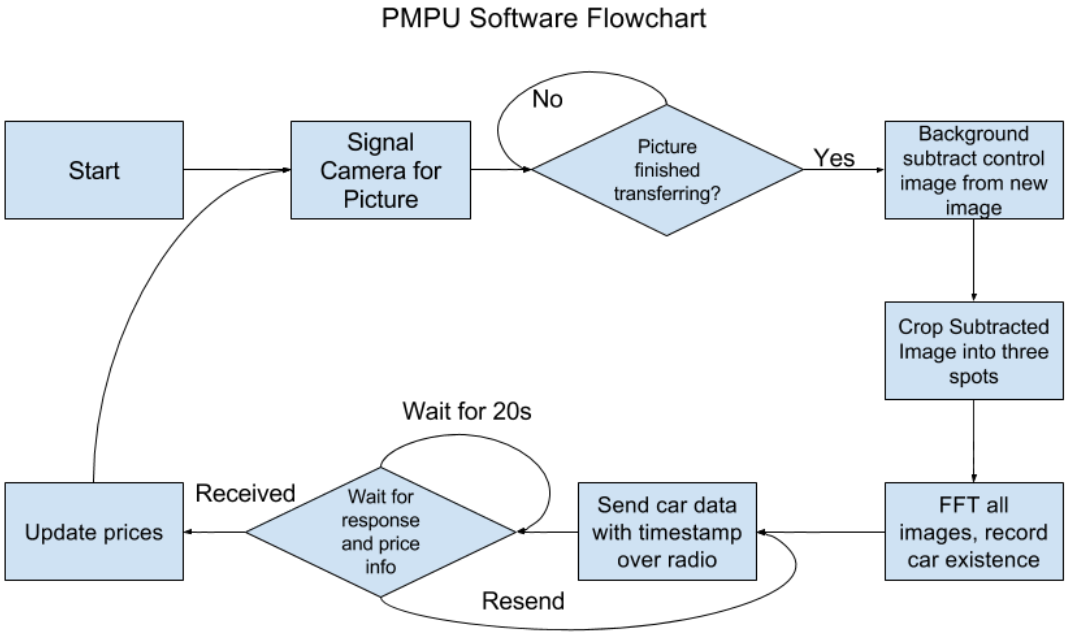


Figure 7: Flowchart for PMPU software

Requirements	Verifications
1. Must be able to store two QQVGA pictures at once 2. Must be able to request a picture from the camera, receive and store it within 1.5 minutes	1.1 Hook PMPU up to laptop and launch provided software 1.2 Request PMPU take and store two pictures 1.3 Request the two stored pictures 1.4 Ensure it returns two QQVGA pictures
2. Must be able to request a picture from the camera, receive and store it within 1.5 minutes	2.1 Hook PMPU up to laptop and launch provided software 2.2 Request PMPU take and store one picture 2.3 Record the time it takes to finish and ensure it is less than 1.5 minutes
3. Must be able to perform a Frame Differencing background subtraction using two separate QQVGA pictures and attenuate the background by 5 dB of the foreground within 1.5 minutes	3.1 Hook PMPU up to laptop and launch provided software 3.2 Load a background and modified image with different foreground into PMPU 3.3 Request it to background subtract 3.4 Ensure it finished in 1.5 minutes 3.4 Request the returned picture 3.5 Verify picture's background has been attenuated by 5 dB
4. Must be able to perform an 8 bit FFT on a picture within 1.5 minutes	4.1 Hook PMPU up to laptop and launch provided software 4.2 Load image into PMPU 4.3 Request it to perform an FFT 4.4 Ensure it finished in 1.5 minutes 4.4 Request the returned spectrum 4.5 Independently run an 8-bit FFT on the image 4.6 Verify that the two spectra are the same
5. Must be able to transmit parking data via Xbee to backend and receive response within 30 seconds.	5.1 Hook PMPU up to laptop and launch provided software 5.2 Make sure that the receiving Xbee attached to the PC running the backend software is in range and running 5.3 Request that the PMPU send parking data 5.4 Check the PC to make sure it received the data 5.5 Ensure it returned in 30 seconds

2.8 Backend Software

This algorithm will control the pricing model that the meters will implement. The majority of the time is spent idling until data is received. Once a packet is received we mark that we are currently in a data collection cycle and identify which meters we've received information from. If a sufficient amount of time has passed (1-2 seconds) without receiving data while in collection mode we check an array to determine which meters we don't have data from and poll them for their data. Once we have everything we record and update our parking density information. If the end of a time slot has been reached we can calculate the new pricing model and send it to the meters.

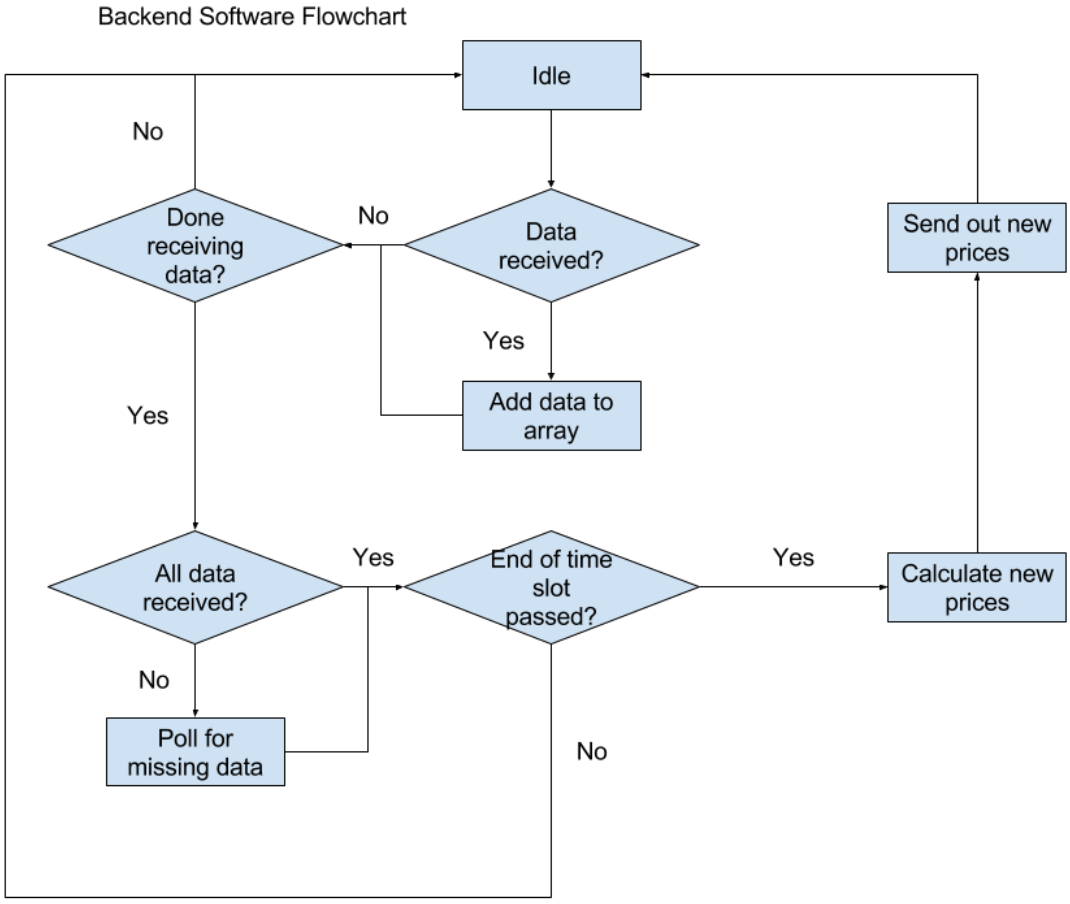


Figure 3: Flowchart for the backend software

Data Collection **(5 Points)**

- Every 5 minutes data is sent from the meter to the backend, when that occurs an ID as well as the parking data will be sent via the transceiver. When it arrives it can be queued and processed later while the system accepts all the other requests. Once requests have stopped for 1 second we can check to see if any requests were missed and poll the missed meters individually.

Requirements	Verification
1. Data from all meters should be collected successfully every 5 minutes. (3 points)	1.1 Set up a data packet and send it to backend using transceiver. 1.2 Confirm it was received. 1.3 Backend can send acknowledge 1.4 Use timing data to establish upper bound on number of meters.
2. If data is missed, it should poll the meters it doesn't have data for. (2 points)	2.1 Repeat process above but have backend randomly ignore some information on first pass 2.2 Have backend print message when all data is received. 2.3 Check array to confirm all data was received.

Data Storage **(2 Points)**

- The backend needs to maintain nine separate arrays: received requests, four pricing models, and four parking density arrays. The "received requests" array will store which requests have been received during the data collection cycle. The pricing model arrays will store the prices to be used for each time slot during the day. The parking density arrays will store the parking densities for the current day as collected from the parking meters. As a result the memory usage per meter is 17 bytes (1 byte for request, 4x4 bytes for pricing).

Requirements	Verification
1. Arrays should contain correct data at correct index. After being received by collection process. (2 points)	1.1 Generate meter data and send it sequentially over transceiver. 1.2 Check array values and confirm they are in the appropriate location

Control Algorithm **(8 Points)**

- We intend to implement a simple proportional-derivative (PD) control algorithm. When all the data for a time slot has been collected it will calculate the new pricing model for that time slot. If the density for an area is too high the price will increase proportionally depending on how far it is from the goal. As the density approaches the goal the price will change slowly to avoid the price "bouncing" from day to day.

Requirements	Verification
1. Changes in prices should reflect the difference between the current density and the goal. If the current density is too high the price should proportionally increase, if it's too low the price should proportionally decrease. (4 points for proportion) (4 points for derivative)	1.1 Setup the algorithm and give it a parking density 1.2 Confirm the correct change happens. 1.3 Set the density close to the goal 1.4 Monitor changes as density changes towards goal to confirm the change decreases.

Simulation System Test **(10 Points)**

- A city simulation will be used as a system test. Personalities will be generated according to a distribution. Each personality will be assigned a location they want to park at, how much they are willing to pay, and a bound on how far away they will park. The bound ensures they will inevitably find a spot to park in, regardless of price. Afterwards parking prices will be adjusted and the personalities will be generated and will attempt to park with the updated pricing model.

Requirements	Verification
1. Generated personalities should resemble the designated distribution with error <10% (3 points)	1.1 Generate 10, 100, 1000 personalities based on distribution. 1.2 Graph personalities and calculate average error.
2. Parking densities should converge and form a stable system within 100 iterations. (1 point for 1x1) (3 points for 5x5) (3 points for 20x20)	2.1 Run simulation for a single grid tile using a park or no park decision 2.2 If this system converges expand to 25 tiles (5x5) 2.3 If this converges expand to 20x20 for final test. Graph densities after at various numbers of iterations (5, 10, 20, 30... etc) to confirm system is converging.

2.9 Circuit Diagram and Preliminary PCB

The circuitry for this project consists of seven major parts, as marked in Figure 6 and shown in Figure 7. Three of these parts are pre-built packages: the ATMega2560, the Arduino TTL Camera, and the Xbee Series 1 transceiver. The other four are small circuits made to support the larger packages and they include: a 5V regulator circuit for powering all three of the large packages, an “On” LED indicator for the ATMega, a reset switch for the ATMega, and a 16MHz oscillator for the ATMega’s clock. Below in Figure 8 is the preliminary PCB for this circuit. Changes need to be made and the Xbee may be moved off-board, but this is preliminarily what it looks like.

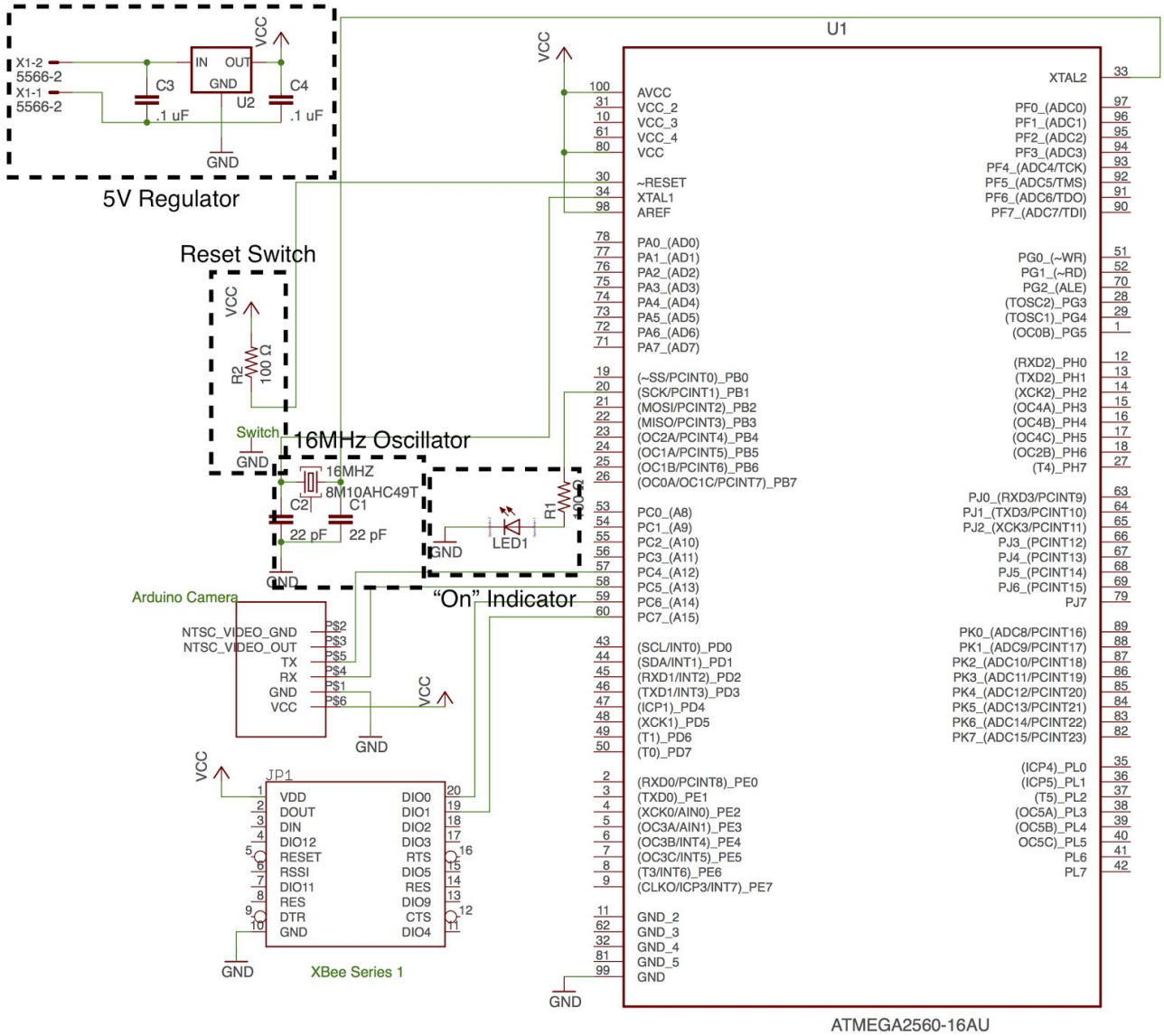


Figure 6: PMPU Circuit Diagram with sections marked

Figure 7: Clean PMPU Circuit Diagram
(Eagle CAD libraries for ATMega [7] and Xbee[8])

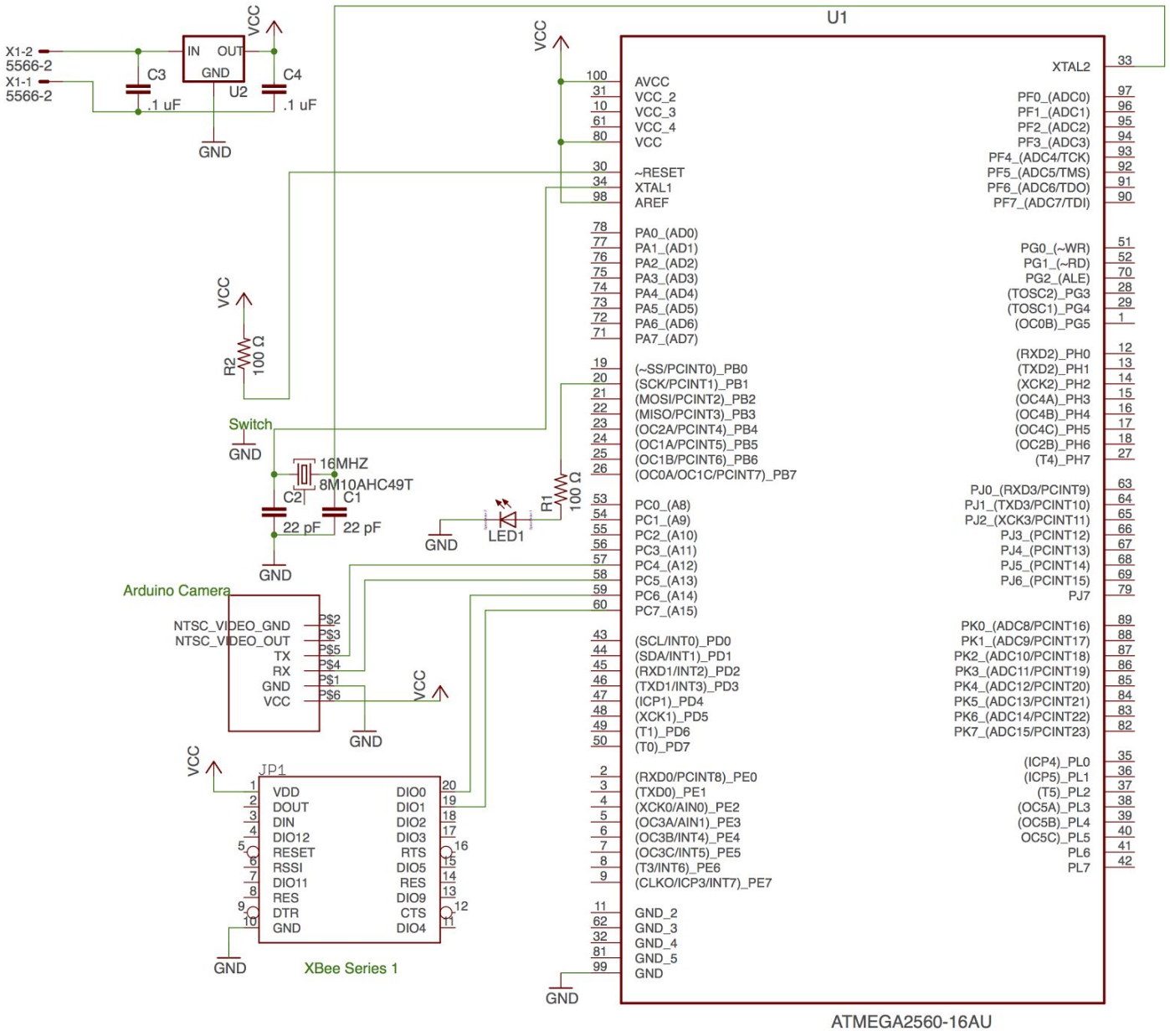
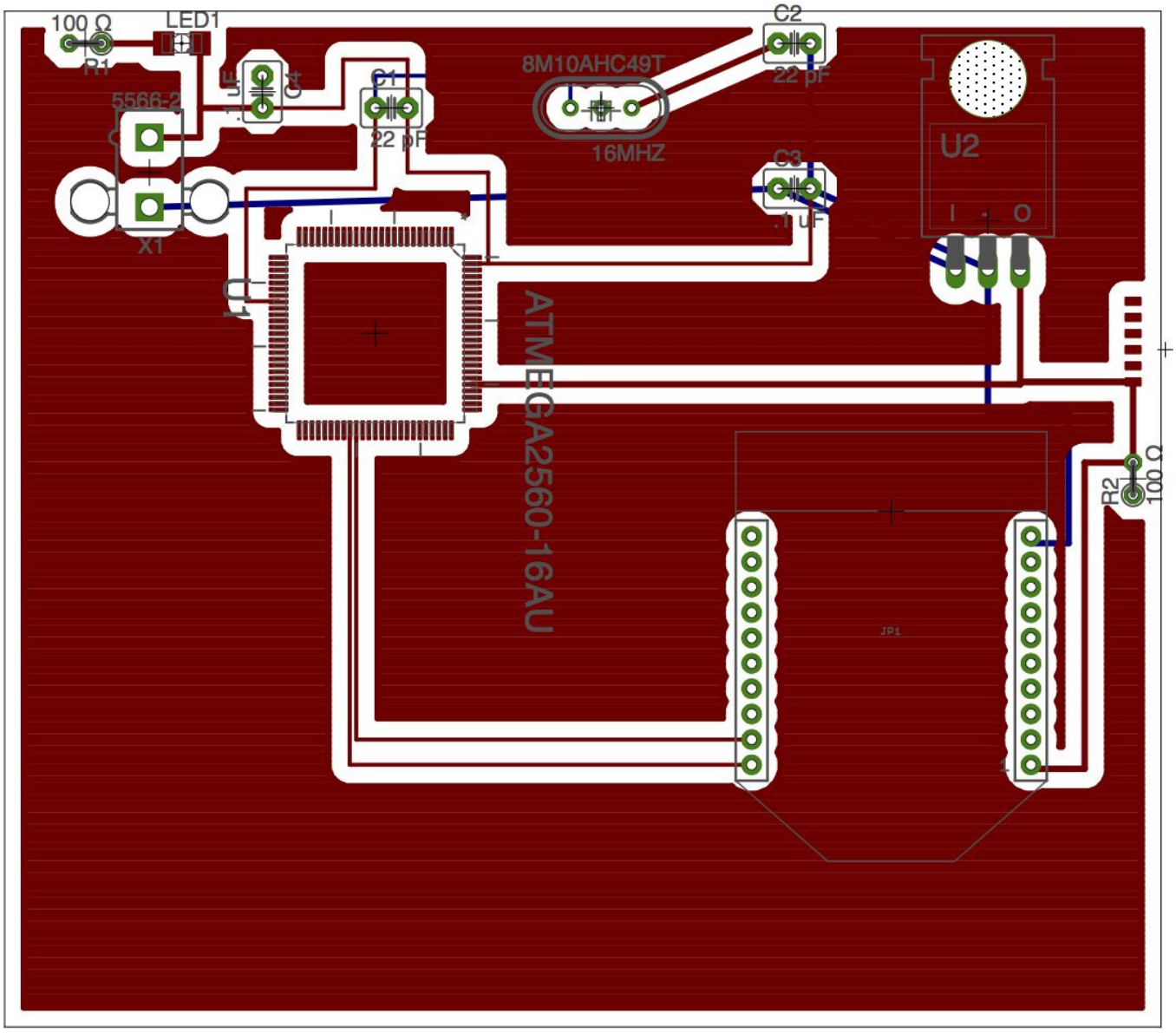


Figure 8: Preliminary PMPU PCB



2.10 Tolerance Analysis

Software Tolerance Analysis

The component that poses the largest risk for project failure is the control algorithm. If the algorithm doesn't work or is ineffective then the prices will adjust improperly. However, a tolerance analysis of this component would require knowing the proportionality constants after tuning the control algorithm, as well as obtaining results from a system test. From there it would be possible to evaluate the tolerance from seeing how many meters providing incorrect information would result in significant error in the pricing model. Thus, a complete tolerance analysis of this component will be completed at a later date.

Hardware Tolerance Analysis

The part of this project that required the most stringent hardware design choices was the oscillator circuit that powers the ATMega2560's internal clock. The main design choices that had to be made was what type of oscillator to use, at which frequency to run it, and which specific oscillator to buy. Looking at the ATMega's datasheet [6] we can see that it accepts frequencies from as low as 2MHz to as high as 16MHz.

Generally, the lower frequency option is available for those who would use the chip in some power constrained environment, such as when using a battery. Since we are not power constrained in this project and need as much processing power as possible, we will be using the maximum frequency of 16MHz.

Next, we had to decide on what type of oscillator to use. There are two main types of oscillators that Atmel recommends for this purpose, crystal oscillators and ceramic oscillators. The main difference between these two type are their *Q* factors and their tuning abilities. *Q* factors are a way of measuring how spread out the frequency they generate is from its center and is calculated by:

$$Q = \frac{f_r}{\Delta f}$$

Where Q is the Q factor, f_r is the resonant frequency of the oscillator and Δf is the Full Width at Half Maximum, or the bandwidth where the oscillation power is at least half the resonant power [9]. Essentially, the larger the Q factor, the more concentrated the bandwidth is and the more regular it will run the ATMega. Atmel suggests the highest Q Factor available for the best operations. However, we also had to consider the oscillator's tuning abilities. Tuning is process of using other components in the circuit to change the resonant frequency of the oscillator and ceramic oscillators can tune much easier than crystals can. This is most useful when needing to dynamically scale processing power, but again, since we will always operate at maximum frequency, we do not need to tune. Due to this and its increased Q factor, we decided to go with a crystal oscillator.

Finally we had to decide which oscillator to buy. Looking at the tolerances for Atmel chips, they run the best when crystal oscillators have a tolerance of less than 400 PPM tolerance [6] Due to this, we decided to go with the 16MHz crystal from adafruit as it has a 300 PPM tolerance [6].

3. Cost and Schedule

3.1 Costs

Costs of Labor

Name	Hourly Rate	Hours Invested	Cost
Nick Johanson	$35	120	$4,200
Adam Barbato	$35	120	$4,200
		Total Labor Cost	$8,400
		Total Labor Cost with Overhead (x2.5)	$21,000

Costs of Parts

Part	Quantity	Unit Cost	Total Cost
ATMega2560	2	$13.25	$26.50
XBee 1mW Trace Antenna (802.15.4)	2	$24.95	$49.90
TTL Serial Camera	1	$54.95	$54.95
Voltage Regulator-5V	2	$0.95	$1.90
16MHz crystal oscillator	1	$0.75	$0.75
Arduino Mega	1	$45.95	$45.95
		Total Cost of Parts	$179.95

The combination of labor and parts results in a total cost for the project of $21,179.95.

3.2 Schedule

Week of:	Person	Responsibility
January 30th	Nick Johanson	**Proposal:** Flow chart, block diagram, and R&V for software.
	Adam Barbato	**Proposal:** Circuit schematic, Safety/Ethics/ and introduction.
February 6th	Nick Johanson	Update software flowchart, break software down into testable modules
	Adam Barbato	Select electronics and design PCB
February 13th	Nick Johanson	Update information from proposal for mock design review
	Adam Barbato	Update information from proposal for mock design review
February 20th	Nick Johanson	Update software flow chart, software R&V, write cost analysis
	Adam Barbato	Update Block diagram, hardware R&V, write tolerance analysis.
February 27th	Nick Johanson	Start coding backend data collection module
	Adam Barbato	Start coding computer vision algorithm on test hardware
March 6th	Nick Johanson	Finalize data collection module and begin testing
	Adam Barbato	Finalize CV Algorithm on test hardware. Ensure it works
March 13th	Nick Johanson	Code data storage module and test
	Adam Barbato	Test hardware on breadboard. Finalize and order PCB
March 20th	Nick Johanson	Spring Break: Work on backlog / start coding simulation
	Adam Barbato	Spring Break: Work on backlog

March 27th	Nick Johanson	Begin testing on simulation for 1x1 grid
	Adam Barbato	Transfer algorithm to Arduino IDE
April 3rd	Nick Johanson	Test simulation using 5x5 and 20x20 grid
	Adam Barbato	Test algorithm with camera and Xbee on test Arduino. Solder PCB together
April 10th	Nick Johanson	Tune control algorithm and run many iterations of simulation
	Adam Barbato	Test on finalized hardware
April 17th	Nick Johanson	Mock Demo
	Adam Barbato	Mock Demo
April 24th	Nick Johanson	Final Demo
	Adam Barbato	Final Demo

4. Ethics and Safety

4.1 Safety

In regards to safety, due to the mundane nature of the project and its goals the only real safety risk during construction would relate to soldering components to a PCB or gathering data while near moving vehicles. In regards to soldering, standard lab safety guidelines which relate to soldering and general lab safety apply here and will be followed and in regards to street safety, standard practices will also apply and be followed.

In a theoretical real world application there are some extended safety concerns. The final version of the meter would need to have a waterproof housing to safeguard against electric shocks and to protect the internal electronics. Additionally, data security would be a crucial component to ensure personal information isn't stolen or abused.

4.2 Ethics

A large ethical issue that has been brought to our attention is the ability to abuse our system to price gouge or discriminate against impoverished people. Any malicious act like this would go directly against the IEEE Code of Ethics' point 1 about keeping the public's welfare in mind and point 8 by discriminating against those who cannot pay [5]. This is absolutely a possibility with our system, as it would be with any system that can control prices of any kind. However, the system was not designed to work that way and changing the system to accomplish either of these unethical goals would be against its main purpose: to make easy parking more available for everybody.

Additionally we will likely be using third party software libraries and it is important for us to credit the authors appropriately by point 7 of the IEEE Code of Ethics [5].

5. Citations

[1] "17.24.050 Parking Facility Layout and Dimensions." 17.24.050 Parking Facility Layout and Dimensions. N.p., n.d. Web. 21 Feb. 2017.

[2] Shoup, Donald C. The High Cost of Free Parking. Chicago: Planners, American Planning Association, 2011. Print.

[3] "Arduino - Setting up an Arduino on a Breadboard." Arduino - Setting up an Arduino on a Breadboard. N.p., n.d. Web. 21 Feb. 2017.

[4] "TTL Serial Camera." Wiring the Camera | TTL Serial Camera | Adafruit Learning System. N.p., n.d. Web. 21 Feb. 2017.

[5] "7.8 IEEE Code of Ethics." IEEE Code of Ethics. IEEE. n.d. Web. 21 Feb. 2017.

[6] “Atmel AVR2067: Crystal Characterization for AVR RF” Atmel Corporation. n.d. Web. 24 Feb 2017

[7] “SparkFun-RF.lbr” SparkFun Eagle Libraries. SparkFun. N.d. Web. 23 Feb. 2017. <https://github.com/sparkfun/SparkFun-Eagle-Libraries/blob/master/SparkFun-RF.lbr>

[8] “Atmel CAD Library for Cadsoft EAGLE Software” Eagle CAD Libraries. element14. N.d. Web. 23 Feb. 2017. <https://www.element14.com/community/docs/DOC-64259/l/atmel-cad-library-for-cadsoft-eagle-software>

[9] "Quality Factor / Q Factor Tutorial." Quality Factor Tutorial | Q Factor Radio-Electronics.com. N.p., n.d. Web. 25 Feb. 2017.