

6-14-2018

B.O.G.G.L.E.S.: Boundary Optical GeoGraphic Lidar Environment System

Miguel Chapa
Santa Clara University

Evan Hoerl
Santa Clara University

Isaac Jorgensen
Santa Clara University

Carl Maggio
Santa Clara University, cmaggio@scu.edu

Follow this and additional works at: https://scholarcommons.scu.edu/idp_senior



Part of the [Computer Engineering Commons](#), and the [Electrical and Computer Engineering Commons](#)

Recommended Citation

Chapa, Miguel; Hoerl, Evan; Jorgensen, Isaac; and Maggio, Carl, "B.O.G.G.L.E.S.: Boundary Optical GeoGraphic Lidar Environment System" (2018). *Interdisciplinary Design Senior Theses*. 36.
https://scholarcommons.scu.edu/idp_senior/36

This Thesis is brought to you for free and open access by the Engineering Senior Theses at Scholar Commons. It has been accepted for inclusion in Interdisciplinary Design Senior Theses by an authorized administrator of Scholar Commons. For more information, please contact rscroggin@scu.edu.

Santa Clara University
DEPARTMENT of COMPUTER ENGINEERING
DEPARTMENT of ELECTRICAL ENGINEERING

Date: June 14, 2018

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY
SUPERVISION BY

Miguel Chapa, Evan Hoerl, Isaac Jorgensen, Carl Maggio

ENTITLED

**B.O.G.G.L.E.S.: Boundary Optical GeoGraphic Lidar Environment
System**

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING
AND
BACHELOR OF SCIENCE IN ELECTRICAL ENGINEERING



THESIS ADVISOR



DATE



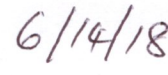
THESIS ADVISOR



DATE



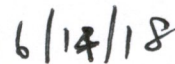
DEPARTMENT CHAIR



DATE



DEPARTMENT CHAIR



DATE

B.O.G.G.L.E.S.: Boundary Optical GeoGraphic Lidar Environment System

by

Miguel Chapa
Evan Hoerl
Isaac Jorgensen
Carl Maggio

SENIOR DESIGN PROJECT REPORT

Submitted in partial fulfillment of the requirements
for the degrees of
Bachelor of Science in Computer Science and Engineering
and
Bachelor of Science in Electrical Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 14, 2018

B.O.G.G.L.E.S: Boundary Optical GeoGraphic Lidar Environment System

Miguel Chapa
Evan Hoerl
Isaac Jorgensen
Carl Maggio

Department of Computer Engineering
Department of Electrical Engineering
Santa Clara University
June 14, 2018

ABSTRACT

The purpose of this paper is to describe a pseudo X-ray vision system that pairs a Lidar scanner with a visualization device. The system as a whole is referred to as B.O.G.G.L.E.S. There are several key factors that went into the development of this system and the background information and design approach are thoroughly described. B.O.G.G.L.E.S functionality is depicted through the use of design constraints and the analysis of test results. Additionally, many possible developments for B.O.G.G.L.E.S are proposed in the paper. This indicates that there are various avenues of improvement for this project that could be implemented in the future.

Table of Contents

1	Acknowledgements	1
2	Introduction	2
3	Additional Background	3
4	Approach	4
4.1	Scanning	4
4.2	Rendering	4
4.3	Visualization	4
4.4	System	5
5	Design Alternatives	6
6	Initial Implementation	7
7	Initial Results	8
8	Revisions	9
8.1	Hardware	9
8.2	Software	9
9	Latest Results	11
10	Significance	12
11	Societal Issues	13
11.1	Ethical	13
11.2	Social	13
11.3	Political	14
11.4	Economic	14
11.5	Health and Safety	14
11.6	Manufacturability	14
11.7	Sustainability	15
11.8	Environmental Impact	15
11.9	Usability	15
11.10	Lifelong Learning	15
11.11	Compassion	15
12	Conclusion	17
	Appendices	19
A	References	19
A.1	Block Diagrams	19
B	Project Logistics	20
B.1	Budget	20
B.2	Development Time Line	20

C	Device Data References	22
C.1	Arduino Uno	23
C.2	Raspberry Pi	27
C.3	Adafruit 16-Channel Servo Driver	32
C.4	Hitec HS422 Servo	59
C.5	Microsoft HoloLens	62
C.6	Adafruit BNO055 Absolute Orientation Sensor	64
C.7	Ultrasonic Ranger Module HC - SR04	91
C.8	Lidar Lite v3	94

List of Figures

1	A model depicting a user, indicated by the red "X", outside of the target environment.	2
2	A model depicting the user's (red "X") view of the target environment with the use of pseudo x-ray vision.	2
3	The system events in the order in which they occur during use of the system.	5
4	Dowd Art & Art History Building, 1st Floor Front Lobby, Scan of 34,000 points in Unity	8
5	Dowd Art & Art History Building, 1st Floor Front Lobby	8
6	Data sheet to be interpreted by Unity, accelerometer data and first data point referenced	10
7	Bannan Engineering, Room 325, Scan in Unity	11
8	Bannan Engineering, Room 325	11
9	Level 0 Block Diagram.	19
10	Level 1 Block Diagram.	19

List of Tables

1	Itemized Budget.	20
2	Development Time Line.	21

1 Acknowledgements

We would first like to thank our advisors, Dr. Sarah Wilson and Dr. Ahmed Amer, for their continued support and unbridled enthusiasm throughout the course of our project. Their excitement for our project rivaled our own and made this whole process not only productive, but also fun. We would also like to thank the SCU Imaginarium for providing us with the hardware resources, including the Microsoft HoloLens, without which this project would not even be possible. Lastly, we would like to thank Microsoft for the use of their developmental resources for such an emerging and up-and-coming technology.

2 Introduction

Everyday, people are placed in life threatening situations. Accidents and disasters occur at a moments notice and require a quick and timely response. We rely on our first responders to keep us safe, but at the potential expense of their lives. In situations where safety is not guaranteed, and minutes or seconds can mean life or death, we want to provide first responders with the best situational awareness and response time to increase the success of their mission. Humans rely greatly on their vision and gain the majority of their environmental awareness from visual cues. In the absence of these visual cues, whether prior to entering or upon entering a space, the risk to the responders life greatly increases. It is our desire to provide first responders with increased spatial awareness in a variety of scenarios by giving them a more informative view of their surroundings.

Current search and rescue methods either put the rescuer in direct danger or completely remove them from the setting using technology such as drones or robots. Entering into a room where the environment is unknown and the risks are unforeseen may place the rescuer and evacuee both in peril. Conversely, removing the rescuer from the environment through sole reliance on video feed prevents the rescuer from naturally visualizing the potentially dangerous environment.

Our solution is to design a proof of concept system that aims to provide the rescuer with an immersive interface to give the user a more natural view of the environment. We will complete this by using real-time room imaging technology to render images that will be viewed using an augmented reality device. We will achieve this by completing three tasks. First, we will create a deployable that will be able to scan a space and relay the data. Second, this information will be processed to create a three dimensional rendering of the area. Finally, this will be ported to a augmented reality device allowing a user in a different space to effectively see into the rendered environment regardless of any obstructions. This will provide the responder with a three dimensional view into the environment, allowing a quick response in an already time sensitive situation. Ideally, our solution will give the responder more information, increasing the probability of a successful rescue.

Our goal is to implement a proof of concept system that gives people the ability to see past physical walls and boundaries. However, the potential does not end there. The successful implementation of this project could someday lead to the visualization of phenomena picked up by our other senses as well. From air pressure to temperature, and invisible gases to sound, the possibilities are endless.

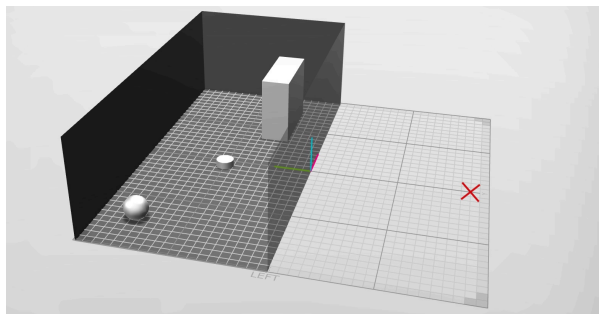


Figure 1: A model depicting a user, indicated by the red "X", outside of the target environment.

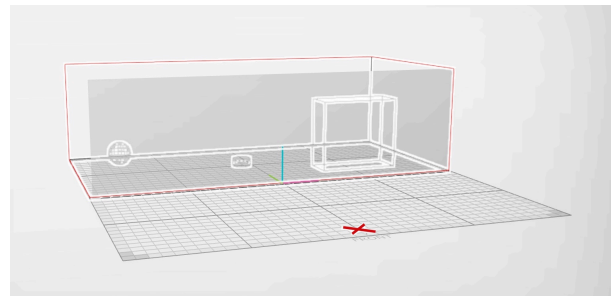


Figure 2: A model depicting the user's (red "X") view of the target environment with the use of pseudo x-ray vision.

3 Additional Background

Our objective for our project was to provide a user with increased spatial awareness by giving them a more informative view of any remote environment. B.O.G.G.L.E.S (Boundary Optical Geo-Graphic Lidar Environment System) was initially designed for search and rescue applications. Additionally in disaster applications it could be beneficial to increase the spatial awareness of the user. In disaster relief efforts, visibility may be reduced due to factors such as smoke and lack of light.

We wanted to implement our system in such a way that the information extracted from a room could be easily visualized without significantly obstructing normal vision. The combination of these key features would yield a pseudo x-ray system that could be used in a variety of applications.

4 Approach

We had several restraints. Our design must:

- be accurate enough that a person could understand the general layout of the room.
- be fast enough to at least show proof-of-concept for real world systems.
- be lightweight and easy to transport.
- be cost effective.
- work in most common search and rescue scenarios, including see through smoke.
- be able to take scans of average size rooms which we found to be 20x20 feet.
- With these requirements in mind, we identified three essential questions to answer prior to beginning production.
 1. How to best obtain the data?
 2. How to best render the data?
 3. How to best visualize the data?

4.1 Scanning

Our first decision was to determine the best method for obtaining the points needed to accurately render the room. We needed a scanner that would allow us to obtain x, y, and z coordinates of points in a space, that still worked within the constraints mentioned above. We settled on Lidar as it could easily scan a 20x20 foot room with a relatively low margin of error. It could measure distance through smoke. Additionally units exist within a reasonable price range. To get the Lidar sensor functioning we used a Raspberry Pi, which is programed in Python, and two Servo motors that would provide us horizontal and vertical rotation.

4.2 Rendering

Our next step was to decide how we were going to render the points provided by our device. We determined this would require utilizing a graphics engine that could easily take data from an outside source, read, interpret, and render it, and finally export the rendering to an mixed reality device. Unity 3D Graphics Engine was the ideal solution as it not only fulfilled all these needs, but it also is very widely used, well documented, and has a low barrier to entry in regards to learning how its various functions work. Unity uses C#, an Object-Oriented language primarily used for application development that is very similar in syntax to C++, which we were all well versed in, and it also has a built in OpenGL graphics library that makes rendering points and other objects very intuitive.

4.3 Visualization

Finally, we determined the best approach to visualize the data. As mentioned, a major issue with current search and rescue methods is that they separate the responder from the target environment. We needed a solution that would still allow for a quick and decisive response without significantly obstructing their view. Using an mixed reality (MR) headset provided an unique answer. It adds computer imagery on top of the real world. It is still a fairly new technology, allowing us to develop a solution distinct from its predecessors. The technology essentially adds a computer screen onto a headset shaped like a set of goggles. We had access to the developers edition of the Microsoft HoloLens, one of the MR headsets publicly available at the time of our project's conception. Unity has built in HoloLens support and tools that would allow us to optimize production.

4.4 System

With these questions answer, we finally had to figure out how each of these sections would then interact with each other to create our fully functioning system. Figure 3 depicts this relationship, with the polar coordinates of each point around the room first taken by the Lidar scanner and sent to Unity. From there, a Unity script renders every points together to create our pointcloud of the desired room, and sends it wirelessly to the Microsoft HoloLens. Finally, the user will be able to move around and interact with the rendered room from with the Mixed Reality Headset.

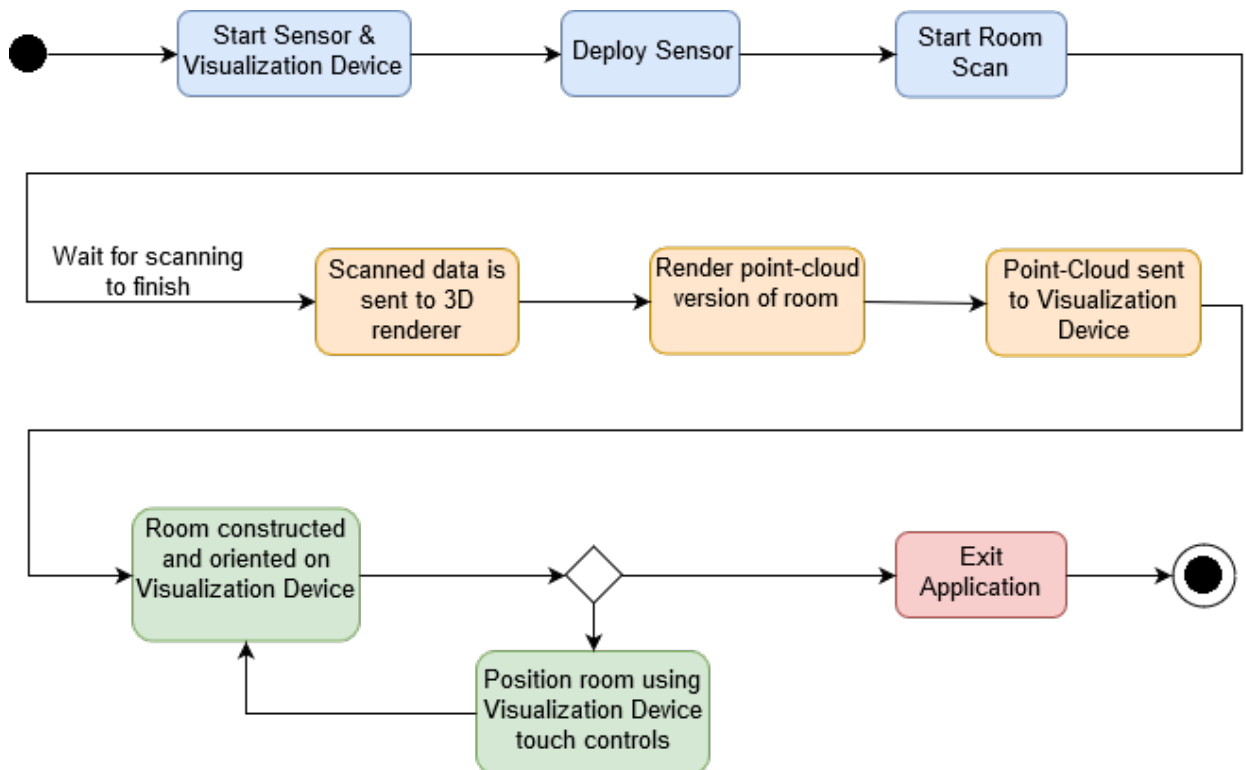


Figure 3: The system events in the order in which they occur during use of the system.

5 Design Alternatives

The first step of our design was to build a sensor that could acquire distance information from a room. Additionally we needed the sensor to acquire distances in areas of low visibility. Since we were focusing on acquiring distances from areas of low visibility we decided not to use a camera for our sensor. The first type of sensor that we considered was sonar. There are various applications in which sonar is used to acquire distance information including seafloor imaging and navigation. After looking at various different components we were able to conclude that the operational range of sonar is too short for our application. Various cost effective sonar sensors have measurement distances that range from 2 to 400 centimeters. This implies that surfaces that are far away from the sensor may not be identified.

Instead of using a sonar sensor for our scanner we decided to use a Lidar sensor. Sonar would require the system to perform potentially taxing computations for every point, and even then, it would still be less accurate than Lidar. While researching various components we decided to use the Lidar Lite v3 by Garmin. This sensor had a range of 40 meters, which is one hundred times larger than the range of the sonar sensors that we had previously considered.

There were a couple alternatives to using the Unity graphics engine. First we could have programmed everything ourselves, without the aid of an engine, and simply downloaded and used the OpenGL graphics libraries by themselves. This would have required more in depth knowledge of how the GPU reads data as well how to develop HoloLens applications from the ground up. This would have prolonged various aspects on the development since Unity provides solutions already built in to its engine. The other alternative would have been using a different graphics engine altogether. The next best option, Unreal Engine, prides itself on adding more complexity to the implementation of projects in return for better looking graphical output. We did not require intensive rendering capability, only the ability to produce points.

6 Initial Implementation

Our project was designed as a proof-of-concept system, which meant the focus was on rapid prototyping with the intent to improve individual areas when needed. Our first step in designing the system was to create a deployable that could accurately and quickly scan a room.

To be able to scan the dimensions of a room we needed the scanner to have a wide range of motion. To implement the motion of the scanner we used two servo motors. (possible insert, Servo Motors w/ directional arrows) The first servo motor was used to rotate the Lidar sensor in the horizontal direction. The second motor was implemented so that it could rotate the Lidar sensor in the vertical direction. The combination of the two servo motors would ensure that the Lidar would get a sweep of the room it was placed in.

Since we used a Lidar sensor and two servo motors in tandem, we needed a microcontroller to dictate the motion and calculations of the deployable scanner. The first microcontroller used was the Arduino Uno. This microcontroller was able to control the two motors simultaneously and extract distance points for our first implementation. Despite having a successful scan of a room, the Arduino Uno was not able to write the data distance points into a text file. Since we could not write directly into a text file, we had to manually transmit the data.

To mitigate this issue we decided to use the Raspberry Pi 3 instead of the Arduino Uno. In order to implement this change we had to change our initial code for the scanner from C to Python. The change in coding language meant that we had to change the structure of the scanners code so that we could still conduct the same scan. Additionally, we had to write into a text file that could be accessed wirelessly. The addition of the Raspberry Pi 3 made it possible to streamline data processes.

Before beginning any software development, we had to determine what methods and tools we would use to interface with the device and develop for the Microsoft HoloLens. Since the HoloLens is still only accessible by developers, there is very little consistent documentation available. As programmers with no mixed reality or HoloLens development experience, we were presented with a confusing task. By drawing on our knowledge of virtual reality development as well as what resources we obtained through Microsoft, we were able to set up our main computer for development. A combination of Unity 3D Graphics Engine and Microsoft Visual Studio was the most documented and best supported setup for HoloLens development. With the addition of the HoloToolKit in Unity as well Visual Studio settings manipulation, the two become easily interfaceable with the HoloLens.

The majority of development took place in Unity. We created scenes and objects to which we could attach scripts. These scripts were written in C#, the standard language for Microsoft development. By applying these scripts to objects in a scene, we could transform them into the point clouds that we desired. We were able to perform testing of the scripts in Unity to analyze the scans. To test the projects on the HoloLens we created executables of the Unity project in Visual Studio that were then transferred to the HoloLens.

7 Initial Results

Upon completing the first iteration of the deployable device, we were able to begin testing by scanning rooms and rendering the data points produced. Our initial test took 24 minutes to complete each scan, which was much too long by our standards. Moving forward, we used a preliminary script which rendered the points in Unity so that we could determine if the point clouds resembled the scanned rooms in any way. We discovered that they did indeed resemble the rooms, but as can be seen when comparing Figure 1, a scan of Santa Clara University's Dowd Art & Art History Building's front lobby, with Figure 2, an actual image of the same front lobby, with only 34,000 points specific details of the room were lost. In order to provide more resolution in the rendering we needed to increase the point density of the point cloud. This would require adjustments with the scanner in an upcoming implementation.

From Unity and Visual Studio we uploaded the executable containing the rendering to the HoloLens to be viewed as a hologram. We immediately noticed that the scaling of the room was too large. Walking around in the hologram of the room failed to provide enough perspective to see what the hologram actually was. In addition, the dispersion of points, as well as the color and brightness, made the point cloud hologram of the room very difficult to see in lighted conditions. Fixing these issues became a top priority moving forward.

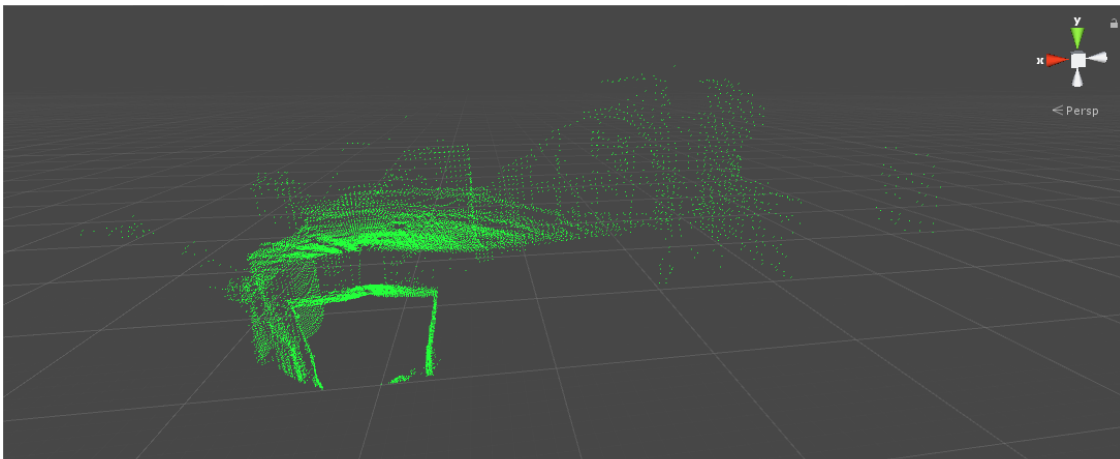


Figure 4: Dowd Art & Art History Building, 1st Floor Front Lobby, Scan of 34,000 points in Unity



Figure 5: Dowd Art & Art History Building, 1st Floor Front Lobby

8 Revisions

8.1 Hardware

One of problems with the Arduino was its inability to run multiple processes. Our initial design only accounted for the Lidar scanner itself and did not account for additional peripherals. We also found that the Arduino IDE would not be robust enough for all of the possible peripheral upgrades we intended on adding. One additional requirement of our new microcontroller was the ability to easily communicate through WiFi or bluetooth as this would better represent the realistic use cases.

Another requirement of our project was the ability for the hologram to be placed in the correct location automatically. The goal was for the hologram to be placed in such a way that the point cloud would align with the actual room. The HoloLens has the ability to track its motion, allowing objects to be anchored relative to the real world. Using this feature, if we can obtain the distance between the HoloLens and the deployable, as well as the respective orientation. There are several methods to obtain this data. We decided to add an accelerometer and gyroscope to the deployable. The user would be able to calibrate the initial location of the deployable relative to the HoloLens and then the deployable would track its location as it was deployed, finally sending back that data once it became stationary.

This works because the gyroscope tracks the rotation of the deployable as it is deployed. This data will be stored as an x, y, and z-axis offsets. This will be sent to the Unity application before the hologram is rendered so it can rotate the hologram correctly. The accelerometer will function similarly in that it will track the acceleration of the deployable over time in the x, y, and z-axis. This data can be integrated over the time to obtain the velocity, and then integrated again to obtain the position. However, accuracy problems may arise since any error of the accelerometer will scale with time squared. This means if the accelerometer data is not accurate then the final positional data could have a significant amount of error.

We also added a dedicated servo controller to act as an intermediate stage between the Raspberry Pi and the motors themselves. The DC motors we used required very specific timed pulses. The Raspberry Pi itself struggles to produce these accurate pulses and we found when trying to run the servos quickly that the accuracy would quickly diminish. By adding a separate servo controller we soon realized that we could run our motors much faster before a noticeable loss in accuracy.

8.2 Software

To address the issues that we discovered in our initial tests we began to make revisions. Before we changed anything else, we had to solve the scaling issue. By introducing a scaling factor to the point rendering script, we were able to shrink the size of the hologram to the real size of the physical room. Accomplishing this allowed us to finally see the rooms details as well as interact with it. In order to interact with it, we implemented tap-to-place functionality through the use of the HoloLens native hand gestures. With tap-to-place now functioning, we were able to pick up and move the hologram at will and align it with the physical room. This supplemented the changes that were made on the deployable, which included adding the gyroscope and accelerometer. Even though the accelerometer data was inaccurate, we still were able to account for the data by interpreting the first three values as an X, Y, and Z distance away from the set origin, as can be seen in Figure 5. Since a single accelerometer was not accurate enough, causing the starting point inaccurate, the tap-to-place functionality was crucial. Hologram visibility still remained an issue however. We remedied this by changing the lighting settings in the Unity scene, increasing both brightness and color intensity of the points.

1.5 4.2 3.0	← Distance from origin (x y z)
0.0 0.0 64.0	← First Data Point (x y z)
0.0 0.0 112.0	
0.0 0.0 112.0	
0.0 0.0 112.0	
0.0 0.0 112.0	
0.0 0.0 114.0	
0.0 0.0 124.0	
0.0 0.0 142.0	
0.0 0.0 148.0	
0.0 0.0 159.0	
0.0 0.0 159.0	
0.0 0.0 159.0	
0.0 0.0 161.0	
0.0 0.0 161.0	
0.0 0.0 161.0	
0.0 0.0 155.0	
0.0 0.0 299.0	
0.0 0.0 369.0	
0.0 0.0 352.0	
0.0 0.0 359.0	
0.0 0.0 388.0	
0.0 0.0 404.0	
0.0 0.0 407.0	
0.0 0.0 407.0	
0.0 0.0 411.0	
0.0 0.0 416.0	
0.0 0.0 416.0	
0.0 0.0 435.0	
0.0 0.0 570.0	
0.0 0.0 549.0	
...	

Figure 6: Data sheet to be interpreted by Unity, accelerometer data and first data point referenced

9 Latest Results

Our latest implementation is the completed proof-of-concept design that we set out to develop. We have created a system that is capable of scanning a remote room using Lidar to produce a point cloud that can be ported into the Microsoft HoloLens for interactive viewing in real space. The scanner must be deployed by hand and placed at an optimal location in the room. Once the scanner has been set, the Lidar sweep of the room is started. It scans at a rate about of 100 degrees/sec and takes just over 5 minutes to complete the full 180 by 90 (@degrees) sweep including the time it takes to reset between each vertical step.

Once complete, the scan (a file containing a list of coordinates in cartesian format) is added to the Unity project which renders the points and compiles into an executable. The program is driven by a script that renders, scales, and reorients the hologram for viewing when the program is run. An executable is created and uploaded to the HoloLens for viewing and future running. This program will remain on the HoloLens for future use in any location. In addition, the hologram renders at a default location when the program starts but may be moved through the use of the HoloLens native touch controls. For easier comprehension of the viewing space, a smaller version of the scanned room is also included in the running application. This sits in the bottom left-hand corner of the HoloLens field of vision. (@possible to elaborate on small room)

Having one accelerometer was not sufficient to accurately measure the displacement of the deployable. It would pick up a large amount of noise which would create an exponentially increasing error as the accelerometer data was twice integrated with respect to the deployables time of flight. There are several ways to improve these results though, such as adding additional accelerometers and averaging their data or taking the median data.

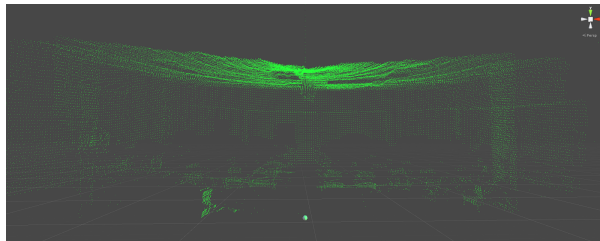


Figure 7: Bannan Engineering, Room 325, Scan in Unity



Figure 8: Bannan Engineering, Room 325

10 Significance

Our objective was to implement a proof-of-concept system that will provide a more informative view into unknown environments, past any physical walls or boundaries. While our system is not quite market ready, we have developed a system that proves this objective is possible. Our full room implementation allows anyone wearing a Mixed Reality headset to view adjacent areas once the Lidar sensor has been deployed and performed its scan. This will be incredibly significant for first responders to gain a better understanding of potentially dangerous environments. With this more informative view, we are increasing the probability of success in these search and rescue situations.

11 Societal Issues

11.1 Ethical

The objective of our design was to create a system that could increase the spatial awareness of users through the use of a scanner and a visualization device. Although there can be a variety of different applications for our design, it was intended to be used by first responders in search and rescue missions. The deployable sensor was intended to be introduced into a foreign environment and it would then scan the room and send the information to the headset. Once the information that the scanner had synthesized had been sent to the visualization device the user would then be able to see the contents and layout of a room without having to physically enter the space.

Our intention was to be able to decrease the potential risk that first responders encounter before they enter any given space in a disaster situation. Although we created our system with good intentions we know that what we have developed is a tool. Tools and their functionality are dictated by their design and the end user. There are numerous ways that our project could be used and not all of these applications are ethical. Since our project is aimed at the acquisition of information it is possible for someone to acquire information in an unethical manner.

Another area in which ethics plays a key role is in our teams conduct and development process. We were able to create our device without breaching the IEEE Code of Ethics. The guidelines are as follows:

- "We, the members of the IEEE, in recognition of the importance of our technologies in affecting the quality of life throughout the world, and in accepting a personal obligation to our profession, its members, and the communities we serve, do hereby commit ourselves to the highest ethical and professional conduct and agree:
 1. to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, and to disclose promptly factors that might endanger the public or the environment;
 2. to avoid real or perceived conflicts of interest whenever possible, and to disclose them to affected parties when they do exist;
 3. to be honest and realistic in stating claims or estimates based on available data;
 4. to reject bribery in all its forms;
 5. to improve the understanding by individuals and society of the capabilities and societal implications of conventional and emerging technologies, including intelligent systems;
 6. to maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations;
 7. to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others;
 8. to treat fairly all persons and to not engage in acts of discrimination based on race, religion, gender, disability, age, national origin, sexual orientation, gender identity, or gender expression;
 9. to avoid injuring others, their property, reputation, or employment by false or malicious action;
 10. to assist colleagues and co-workers in their professional development and to support them in following this code of ethics."

11.2 Social

The goal of this project was to design an innovative system that would have a positive impact in society. The initial idea for this project was conceived immediately after the earthquakes that occurred in Mexico in 2017. The problem that we saw in that situation is that rescue workers needed to risk their lives in order to be able to come to the aid of others. Our system was devised to provide rescue workers with more information of rooms that they did not

have access to. Ideally, our design would not only reduce the risk that reduce workers face in emergency situations but also increase the success rate of certain tasks.

The situation in Mexico served as an inspiration for us to design B.O.G.G.L.E.S. We did not design it specifically for the earthquakes that occurred but rather for a variety of different search and rescue applications. The key social issue that we targeted for the duration of our project was improving the innovating the technology revolved around search and rescue. Disaster situations occur throughout the world and creating an innovative system like B.O.G.G.L.E.S. would be one way of mitigating disaster relief efforts globally.

11.3 Political

Since our project is a private endeavor rather than a public one, there is not much direct political impact to be considered. Any political impact that it may have would be indirect through the way in which it is used or the tasks for which it is used. If wielded as intended, it will only benefit society. Over all, it is very unlikely that anything regarding our device would end up on a public ballot.

11.4 Economic

A key feature of our project was to create a deployable sensor prototype using cost-effective components. The reasoning behind this was not only because we had a \$751.00 dollar budget but also because we wanted to be able to produce a cheaper Lidar scanner than what is currently available on the market. Given the budgetary restraints that we had we were able to produce an effective prototype for our sensor. The final prototype that we produced cost \$275.66 to build. We needed to implement some different hardware to get to our final stage and this meant that we needed to spend more of our budget acquiring new parts. If mass production were ever implemented for this device the cost per unit would see a decrease.

Another economic factor that would need to be considered for B.O.G.G.L.E.S. implementation is the acquisition of a mixed reality headset. For our project we were able to acquire the headset free of charge. However, in order to fully consider the economic implications of our design we must incorporate the cost of the headset. The headset that was used for our system was the Microsoft HoloLens. This developer kit is currently valued at \$3000. This price point makes B.O.G.G.L.E.S. more expensive as a whole. Something to note is that mixed reality is an emerging technology and various companies are on track to release headsets in the coming years. With an increase in competitors there is bound to be a significant decrease in the price of consumer ready mixed reality headsets.

11.5 Health and Safety

B.O.G.G.L.E.S is currently a proof-of-concept system and should not be commercialized in its current state. Before this system is commercialized there needs to be a way to measure the percent error associated with the point cloud that is extracted by the Lidar sensor. If this device were to be released without reliability testing then there would be potential for user injury. The B.O.G.G.L.E.S user depends on the information on the headset to be correct so that they can make informed decisions in disaster situations. The presence of false or incorrect information being sent from the Lidar sensor to the headset could prove to be detrimental to the user.

11.6 Manufacturability

The system that we have developed is software with an associated deployable that is to be used in conjunction with a mixed reality headset. We have developed this system with the expectation that the user already has said headset. If that is not the case, purchasing the mixed reality headset is the largest investment required for implementing our

system. However, new headsets are coming out more and more frequently at continually cheaper costs. The rest of our system is manufacturable at a cheaper cost than pre-existing room/environment scanners. The most likely problem that may arise is the potential rise in cost of the scanner as it is improved in the future.

11.7 Sustainability

The device that we have created to supplement the headset and software is meant to be reused. If any component of the device was to incur damage, it could be replaced without having to replace the whole device. This modular design prevents waste and promotes reusability of still functioning parts. In addition, our device uses parts that can be recycled through an electronic recycling facility.

11.8 Environmental Impact

The main environmental impact of our project comes from the physical components we use. Our system uses a Lidar scanner atop two servo motors as well as a microcontroller and the casing for the system. The current motors we are using are not essential to the design as almost any type of motors will suffice so the environmental impact will need to be looked at on a case-by-case basis. Because selection of motors is so large though, it is possible to take into account environmental impact as a factor when deciding which motor to use ensuring there are no extreme negative impacts. The next area of concern is the microcontroller. The microcontroller will be able to be recycled like all computer parts and has a very low environmental impact. The most fundamental component of our system is the Lidar sensor. High-end Lidar sensors use a very precise photodetector most commonly made of Indium Gallium Arsenide. This is not a naturally occurring compound and it requires vapor decomposition to be used in its formation. This causes the release of Arsenic Hydroxide which is a very toxic gas.

11.9 Usability

In its current state as a proof-of-concept system, set-up and use requires some knowledge of how the system is set up. Our next iteration of the system includes streamlining the scanning and rendering process to eliminate the computer middle-man. Once this is implemented along with a full-fledged GUI, the system will be simple and usable for any individual, including first responders.

11.10 Lifelong Learning

Our project consisted mostly of learning and implementing new skills. Despite having electrical and computer engineering backgrounds, none of the individuals in our group have taken classes or received explicit training for any of the software or hardware we interacted with and created. This required a lot of self-teaching on our parts. Troubleshooting forced us to learn the most while trying to solve obscure issues with no clearly identifiable solution.

11.11 Compassion

B.O.G.G.L.E.S. addresses compassion in an indirect way. If used in the capacity which we created it for, it will allow first responders to save lives and protect themselves from potential harm. Any event, such as an earthquake or a fire, may create hazardous situations for rescue workers trying to reach trapped and frightened victims. With the help of B.O.G.G.L.E.S., not only will the chance of rescuing the victims increase, but the first responders will be able to do so with significantly less risk to themselves. Saving human lives will prevent much suffering for those who

would be left behind. The compassion we feel for those who are at risk of being placed in such dangerous scenarios is what will drive us to ensure that this system is implemented and made available in the near future.

12 Conclusion

The objective of our project was to provide the user with increased spatial awareness by providing a more informative view of any remote environment. In order to do this we needed to develop a system that would emulate pseudo x-ray vision by creating a room scanner and pairing that with a visualization device/headset. Once paired, the data points from the scanner would produce a point cloud that would be sent to the headset. Once the point cloud is sent to the headset, the user would have a general idea of what the room contains as well as its size.

The first step of our design was to create a sensor that could extract distances from a room and write those distances into a file that could be accessed by the headset. In our design we were able to implement a fully functional Lidar sensor that could send point data to the headset via text file.

Using Unity 3D Graphics Engine, we developed a program that would render and manipulate the point cloud of the scanned room in the viewing environment. That program was compiled in Visual Studio as an executable that could be transferred to the HoloLens and ran. The point cloud rendering of the room could then be viewed through the HoloLens as a hologram. The hologram is interactable in the sense that a user can walk through and around it as well as pick it up and move its location.

B.O.G.G.L.E.S is a unique proof-of-concept system that addresses a relevant problem in society. We designed our system for search and rescue missions and the prototype that we produced was extremely innovative. Our final prototype was quite successful and met some of the requirements that we wanted to fulfill. Additionally, we were able to interface the various different components and create a fully functional system.

Despite fulfilling the our initial requirements, there were certain features that we wanted to incorporate into our system that we did not complete or were not in our scope of work. The first feature that we wanted to keep on developing was the complete incorporation of the gyroscope and accelerometer into the deployable scanner. This addition would help us effectively address the position and orientation of the rendered room.

Another feature that we would have liked to build upon was the scan time of the deployable sensor. In our best implementation our fastest scan was roughly five minutes long. In order to improve this time there would need to be some implementation in which we used more effective servo motors. Being able to use faster motors would make the scan duration shorter and increase the effectiveness of our deployable.

For the software end, B.O.G.G.L.E.S needs a fully functional application with an interactive graphic user interface. This would not only make our system more user friendly, but it would also imply that the headset and deployable sensor directly interact with each other. This direct interaction would eliminate the need for a computer to render the room.

For future applications we would want B.O.G.G.L.E.S to be compatible with several different headsets. In our scope of work we focused on pairing the deployable sensor with the Microsoft HoloLens. Many other headsets are going to be released by various companies in the near future so developing a system that can be easily compatible with different headsets could make B.O.G.G.L.E.S a more accessible product. With developments that are being made by upcoming Mixed Reality headsets it would be wise to incorporate more interactivity between the user and the room scans developed by the sensor. Companies such as Magic Leap and Meta are working on headsets that will have more intuitive motion tracking for the user to invoke. Being able to use these more developed motions will increase the interactivity that the user will have with the rendered space.

Fantastical ideas are not impossible and out of reach. We sought out to create a pseudo x-ray vision using holograms. This is something that most people, including ourselves, normally associate with science fiction. Yet we were able to make it a reality. Throughout this process, we discovered that there are many implementations and variations of our project that could be created.

From a technical aspect we learned a lot about the devices that we used as well as augmented and mixed reality technology. Understanding how Lidar works, and specifically, how our Lidar worked with the libraries that ran it, let us to drastically increase the speed of our scans. We learned about the various capabilities of microcontrollers, which included how we could best leverage those capabilities to add features to our device. The largest learning curve

that we came across was encountered while developing for the Microsoft HoloLens. This required a large amount of background reading and development practice so that we could begin developing on our own. There are many new things to consider and understand when working with the HoloLens because it is the only mixed reality device on the market. Understanding the features of the HoloLens was necessary before we could even begin development. With this knowledge, we are better prepared to expand our designs beyond the initial objectives we set out to achieve.

Appendices

A References

A.1 Block Diagrams

Figures 2 and 3 are both our level 0 and level 1 block diagrams outlining the principal functions of our system. The Level 0 Diagram explains a higher level design process of our initial project model. The Level 1 Diagram delves deeper into the processes of our design and depicts the intricacies of our implementation.

1. Level 0 Diagram

This diagram shows the direct chain of data starting from the Lidar up until it is rendered in the HoloLens.

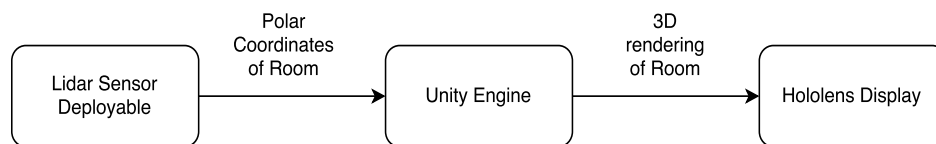


Figure 9: Level 0 Block Diagram.

2. Level 1 Diagram

This diagram shows the direct chain of data starting from the Lidar up until it is rendered in the HoloLens, as well as more specific functionality of the Lidar deployable

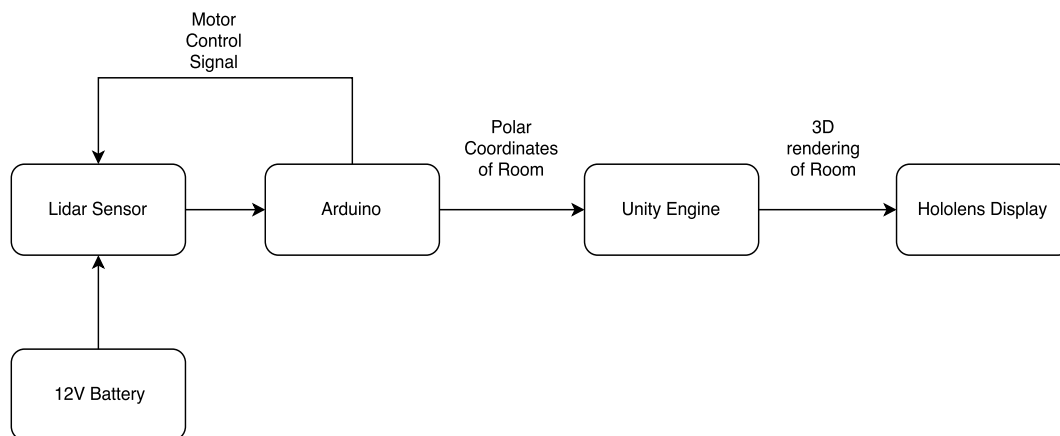


Figure 10: Level 1 Block Diagram.

B Project Logistics

B.1 Budget

Component	Cost
Lidar Lite	\$129.99
Raspberry Pi 3	\$42.99
HS-422 Servos	\$22.98
Pan and Tilt Kit	\$29.93
Adafruit-BNO055 Gyroscope	\$32.27
Adafruit 16 Channel PWM/ Servo Driver	\$17.50
TOTAL	\$275.66

Table 1: Itemized Budget.

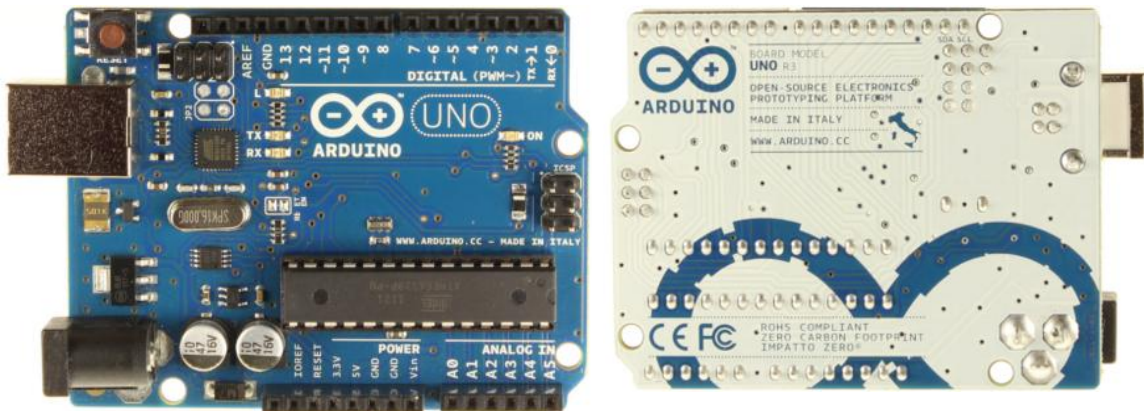
B.2 Development Time Line

In order to organize the design and implementation of our project, we have developed a time line, shown in Table 2. This time line outlines our main tasks throughout fall, winter, and spring quarters. The colors are used to show each groups tasks.

C Device Data References

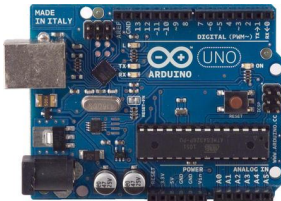
C.1 Arduino Uno

Arduino Uno



Arduino Uno R3 Front

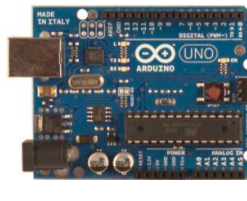
Arduino Uno R3 Back



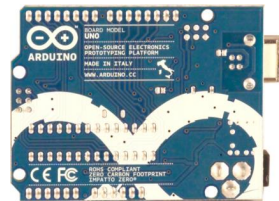
Arduino Uno R2 Front



Arduino Uno SMD



Arduino Uno Front



Arduino Uno Back

Overview

The Arduino Uno is a microcontroller board based on the ATmega328 ([datasheet](#)). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

[Revision 2](#) of the Uno board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into DFU mode.

[Revision 3](#) of the board has the following new features:

- 1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with 3.3V. The second one is a not connected pin, that is reserved for future purposes.
- Stronger RESET circuit.
- Atmega 16U2 replace the 8U2.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the [index of Arduino boards](#).

Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V

Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Schematic & Reference Design

EAGLE files: [arduino-uno-Rev3-reference-design.zip](#) (NOTE: works with Eagle 6.0 and newer)

Schematic: [arduino-uno-Rev3-schematic.pdf](#)

Note: The Arduino reference design can use an Atmega8, 168, or 328, Current models use an ATmega328, but an Atmega8 is shown in the schematic for reference. The pin configuration is identical on all three processors.

Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the [EEPROM library](#)).

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using [pinMode\(\)](#), [digitalWrite\(\)](#), and [digitalRead\(\)](#) functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the [attachInterrupt\(\)](#) function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the [analogWrite\(\)](#) function.

- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication using the [SPI library](#).
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though it is possible to change the upper end of their range using the AREF pin and the [analogReference\(\)](#) function. Additionally, some pins have specialized functionality:

- **TWI: A4 or SDA pin and A5 or SCL pin.** Support TWI communication using the [Wire library](#).

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with [analogReference\(\)](#).
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the [mapping between Arduino pins and ATmega328 ports](#). The mapping for the ATmega8, 168, and 328 is identical.

Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, [on Windows, a .inf file is required](#). The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A [SoftwareSerial library](#) allows for serial communication on any of the Uno's digital pins.

The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the [documentation](#) for details. For SPI communication, use the [SPI library](#).

Programming

The Arduino Uno can be programmed with the Arduino software ([download](#)). Select "Arduino Uno" from the **Tools > Board** menu (according to the microcontroller on your board). For details, see the [reference](#) and [tutorials](#).

The ATmega328 on the Arduino Uno comes preburned with a [bootloader](#) that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol ([reference](#), [C header files](#)).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see [these instructions](#) for details.

The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available. The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use [Atmel's FLIP software](#) (Windows) or the [DFU programmer](#) (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See [this user-contributed tutorial](#) for more information.

Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload. This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

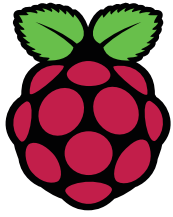
The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see [this forum thread](#) for details.

USB Overcurrent Protection

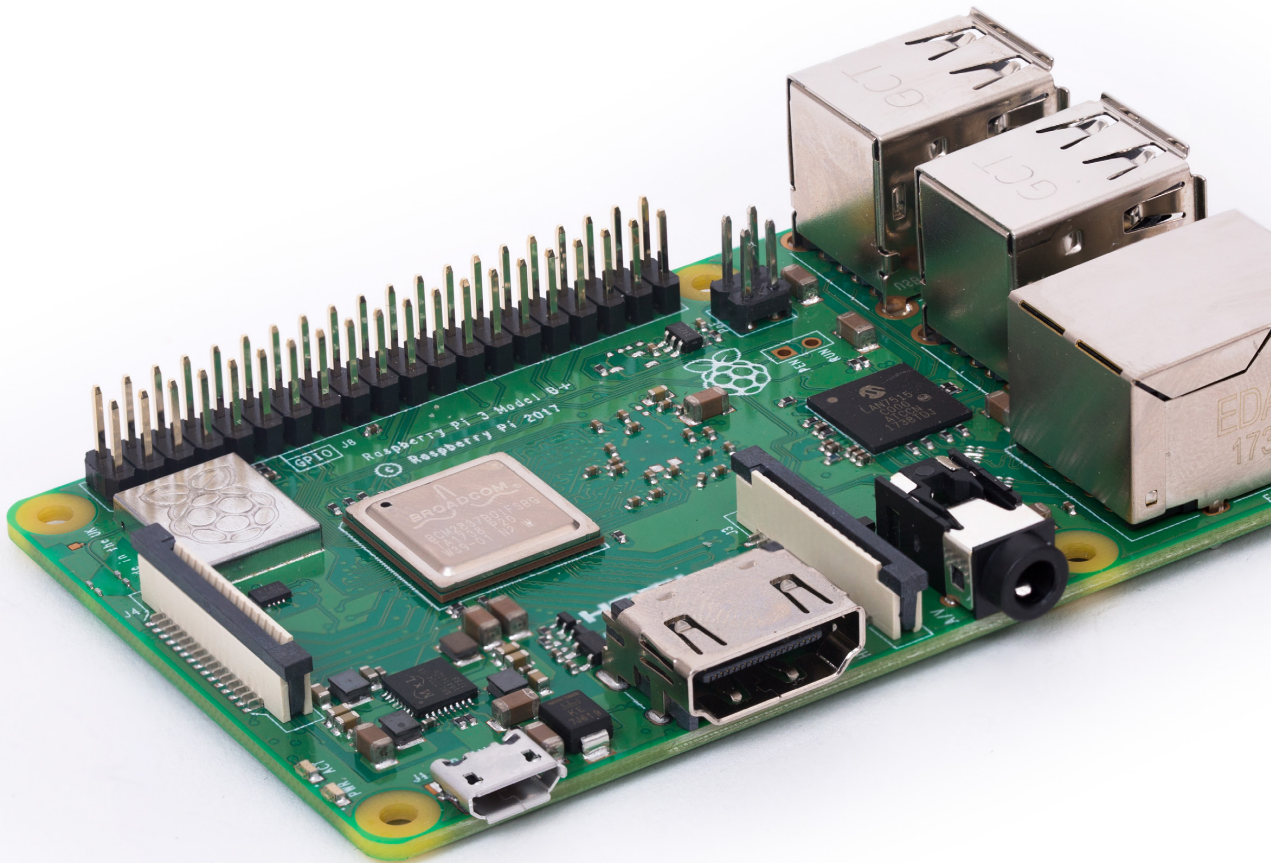
The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics

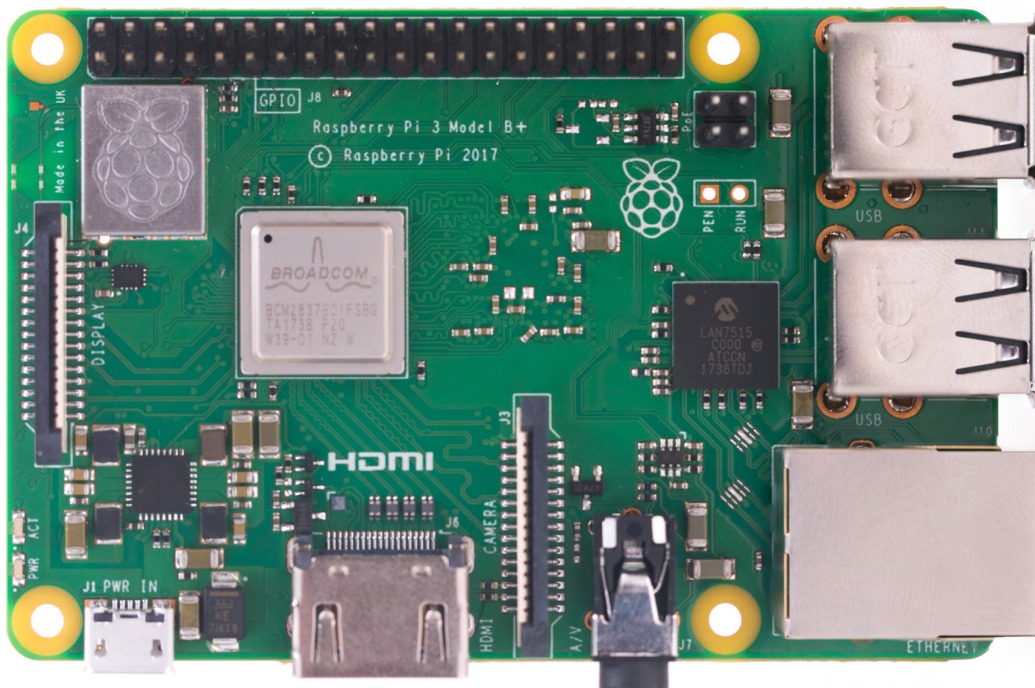
The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.



Raspberry Pi 3 Model B+



Overview



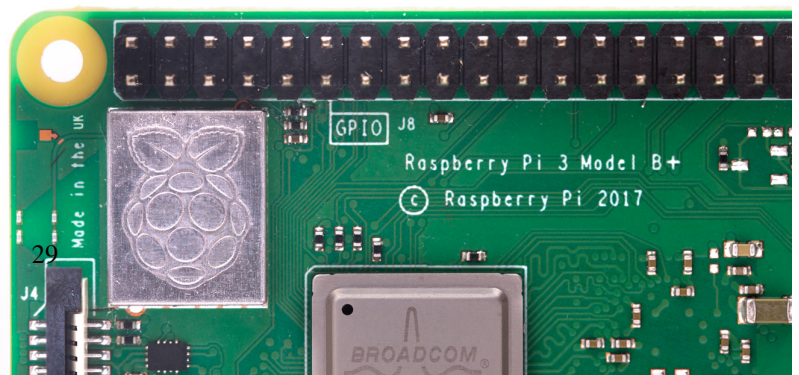
The Raspberry Pi 3 Model B+ is the latest product in the Raspberry Pi 3 range, boasting a 64-bit quad core processor running at 1.4GHz, dual-band 2.4GHz and 5GHz wireless LAN, Bluetooth 4.2/BLE, faster Ethernet, and PoE capability via a separate PoE HAT

The dual-band wireless LAN comes with modular compliance certification, allowing the board to be designed into end products with significantly reduced wireless LAN compliance testing, improving both cost and time to market.

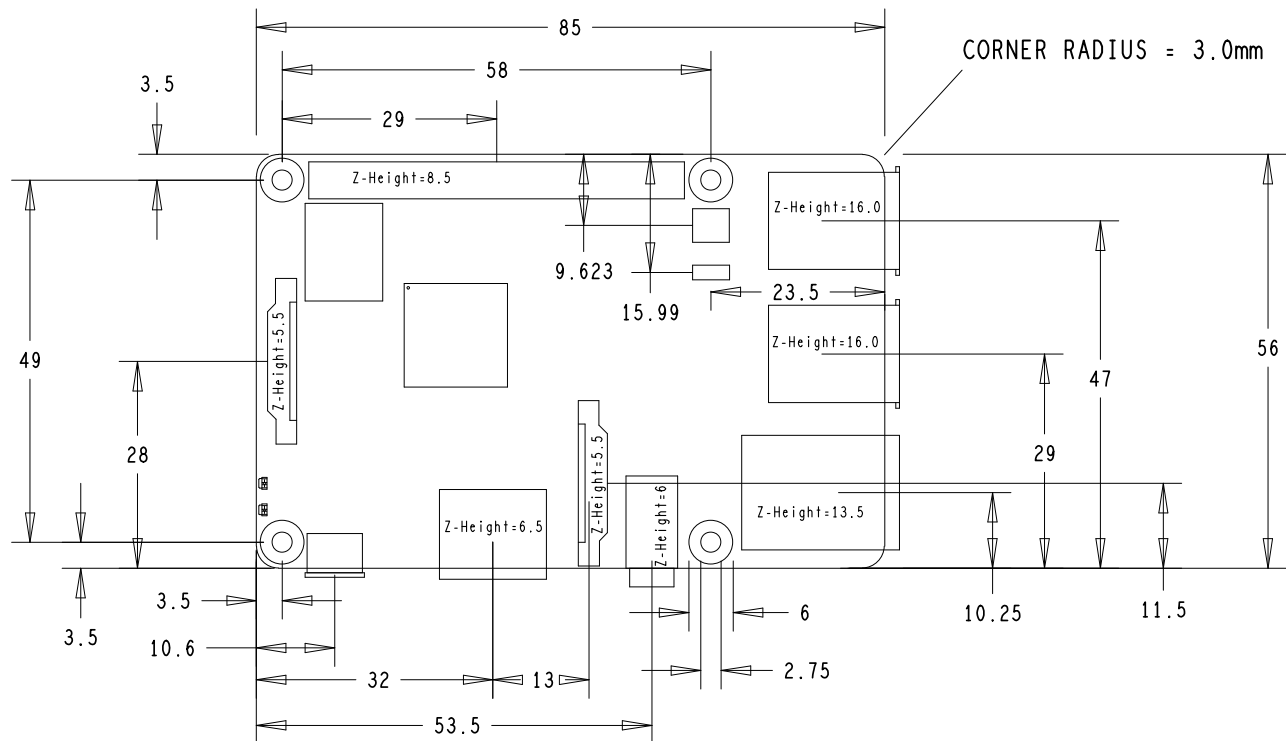
The Raspberry Pi 3 Model B+ maintains the same mechanical footprint as both the Raspberry Pi 2 Model B and the Raspberry Pi 3 Model B.

Specifications

Processor:	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
Memory:	1GB LPDDR2 SDRAM
Connectivity:	<ul style="list-style-type: none">■ 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE■ Gigabit Ethernet over USB 2.0 (maximum throughput 300Mbps)■ 4 × USB 2.0 ports
Access:	Extended 40-pin GPIO header
Video & sound:	<ul style="list-style-type: none">■ 1 × full size HDMI■ MIPI DSI display port■ MIPI CSI camera port■ 4 pole stereo output and composite video port
Multimedia:	H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics
SD card support:	Micro SD format for loading operating system and data storage
Input power:	<ul style="list-style-type: none">■ 5V/2.5A DC via micro USB connector■ 5V DC via GPIO header■ Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
Environment:	Operating temperature, 0–50 °C
Compliance:	For a full list of local and regional product approvals, please visit www.raspberrypi.org/products/raspberry-pi-3-model-b+
Production lifetime:	The Raspberry Pi 3 Model B+ will remain in production until at least January 2023.



Physical specifications



Warnings

- This product should only be connected to an external power supply rated at 5V/2.5A DC. Any external power supply used with the Raspberry Pi 3 Model B+ shall comply with relevant regulations and standards applicable in the country of intended use.
- This product should be operated in a well-ventilated environment and, if used inside a case, the case should not be covered.
- Whilst in use, this product should be placed on a stable, flat, non-conductive surface and should not be contacted by conductive items.
- The connection of incompatible devices to the GPIO connection may affect compliance, result in damage to the unit, and invalidate the warranty.
- All peripherals used with this product should comply with relevant standards for the country of use and be marked accordingly to ensure that safety and performance requirements are met. These articles include but are not limited to keyboards, monitors, and mice when used in conjunction with the Raspberry Pi.
- The cables and connectors of all peripherals used with this product must have adequate insulation so that relevant safety requirements are met.

Safety instructions

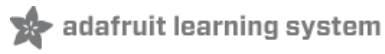
To avoid malfunction of or damage to this product, please observe the following:

- Do not expose to water or moisture, or place on a conductive surface whilst in operation.
- Do not expose to heat from any source; the Raspberry Pi 3 Model B+ is designed for reliable operation at normal ambient temperatures.
- Take care whilst handling to avoid mechanical or electrical damage to the printed circuit board and connectors.
- Whilst it is powered, avoid handling the printed circuit board, or only handle it by the edges to minimise the risk of electrostatic discharge damage.



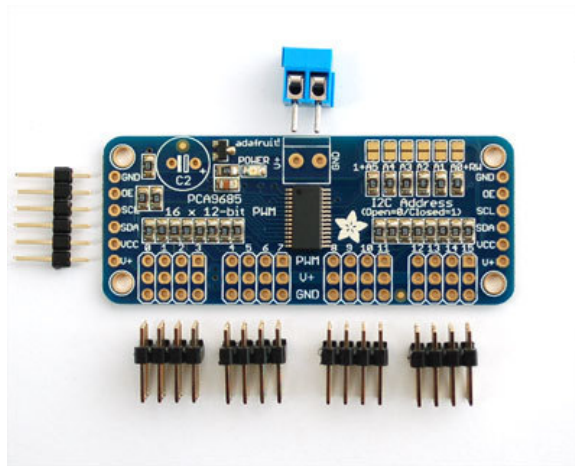


C.3 Adafruit 16-Channel Servo Driver



Adafruit 16-Channel Servo Driver with Arduino

Created by Bill Earl



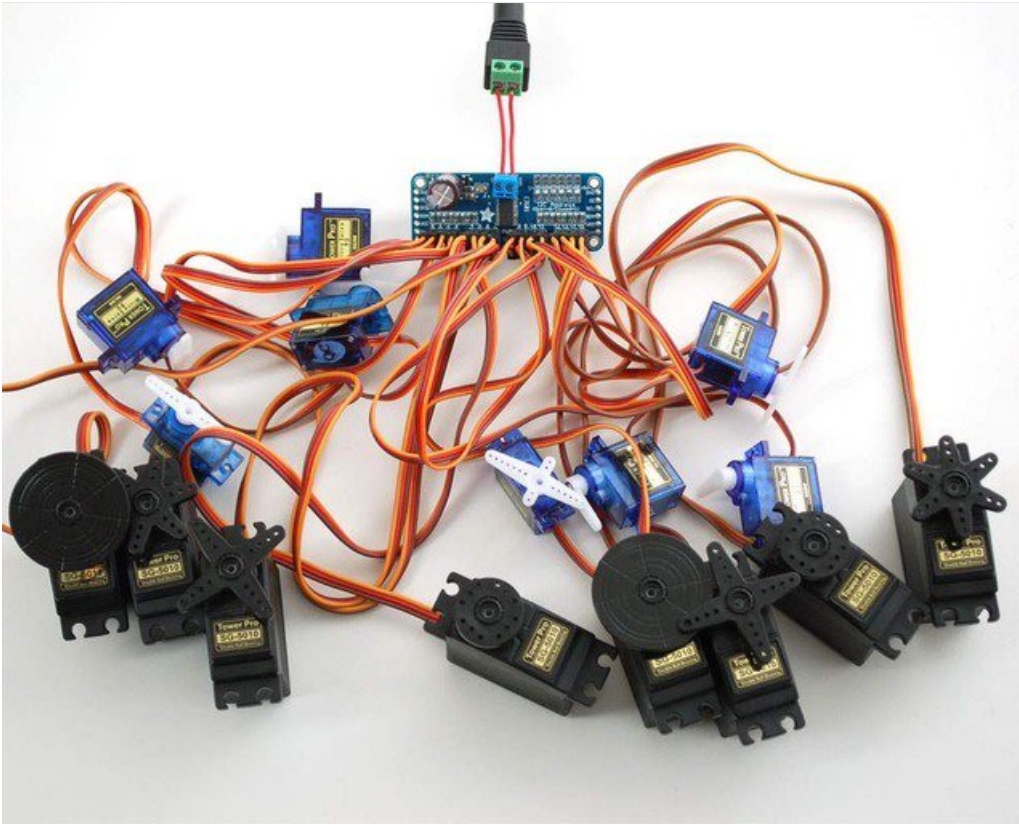
Last updated on 2018-01-16 12:17:12 AM UTC

Guide Contents

Guide Contents	2
Overview	4
Pinouts	6
Power Pins	6
Control Pins	6
Output Ports	6
Assembly	8
Install the Servo Headers	8
Solder all pins	8
Add Headers for Control	8
Install Power Terminals	9
Hooking it Up	10
Connecting to the Arduino	10
Power for the Servos	10
Adding a Capacitor to the thru-hole capacitor slot	11
Connecting a Servo	11
Adding More Servos	12
Chaining Drivers	13
Addressing the Boards	13
Using the Adafruit Library	15
Install Adafruit PCA9685 library	15
Test with the Example Code:	15
If using a Breakout:	16
If using a Shield:	16
If using a FeatherWing:	16
Connect a Servo	16
Calibrating your Servos	16
Converting from Degrees to Pulse Length	17
Library Reference	18
setPWMFreq(freq)	18
Description	18
Arguments	18
Example	18
setPWM(channel, on, off)	18
Description	18
Arguments	18
Example	18
Using as GPIO	18
Arduino Library Docs	20
CircuitPython	21
Adafruit CircuitPython Module Install	21
Bundle Install	21
Usage	22
I2C Initialization	22

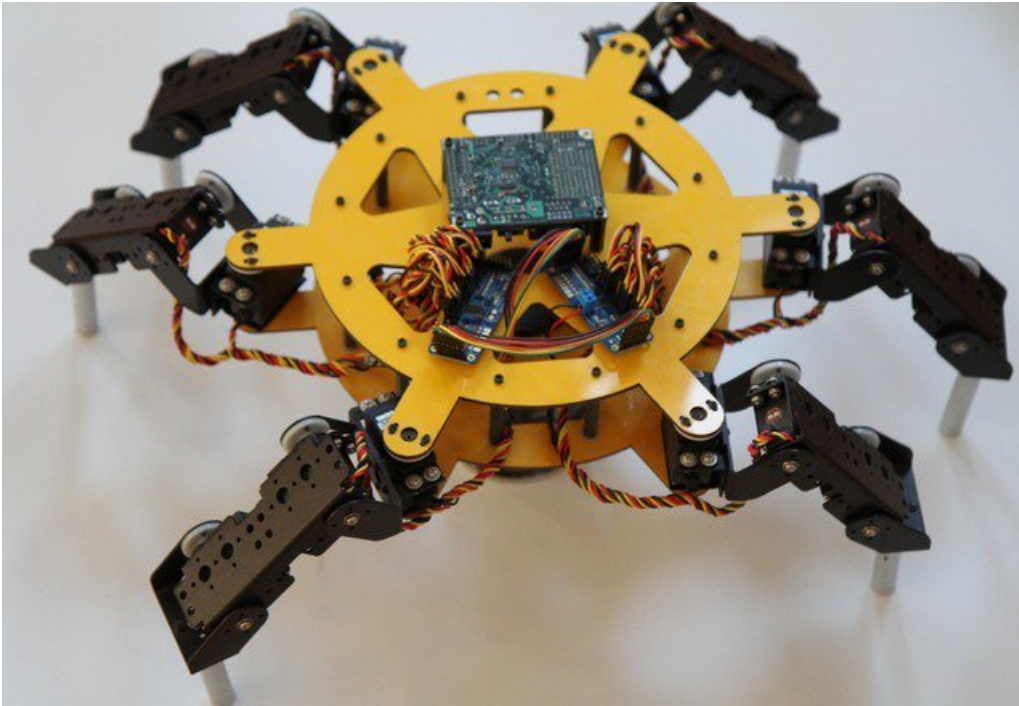
Dimming LED's	22
Control Servos	24
Downloads	27
Files	27
Schematic & Fabrication Print	27
FAQ	29
Can this board be used for LEDs or just servos?	29
I am having strange problems when combining this shield with the Adafruit LED Matrix/7Seg Backpacks	29
With LEDs, how come I cant get the LEDs to turn completely off?	29

Overview

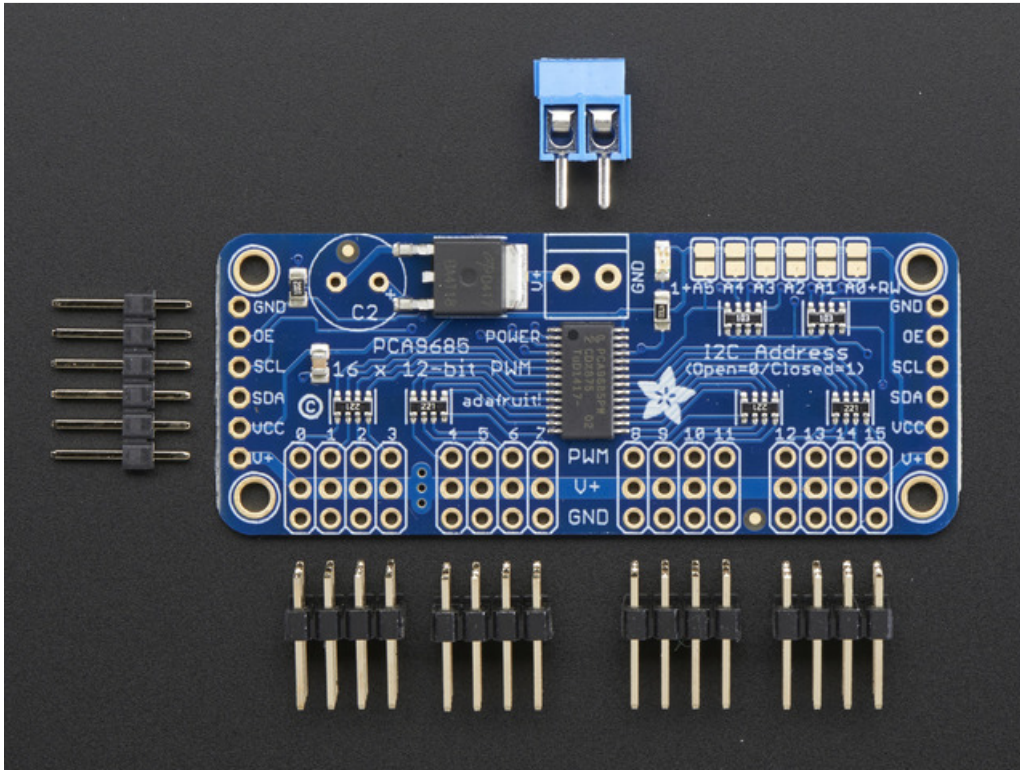


Driving servo motors with the Arduino Servo library is pretty easy, but each one consumes a precious pin - not to mention some Arduino processing power. The Adafruit 16-Channel 12-bit PWM/Servo Driver will drive up to 16 servos over I2C with only 2 pins. The on-board PWM controller will drive all 16 channels simultaneously with no additional Arduino processing overhead. What's more, you can chain up to 62 of them to control up to 992 servos - all with the same 2 pins!

The Adafruit PWM/Servo Driver is the perfect solution for any project that requires a lot of servos.



Pinouts



There are two sets of control input pins on either side. **Both sides of the pins are identical!** Use whichever side you like, you can also easily chain by connecting up two side-by-side

Power Pins

- **GND** - This is the power and signal ground pin, must be connected
- **VCC** - This is the **logic** power pin, connect this to the logic level you want to use for the PCA9685 output, should be 3 - 5V max! It's also used for the 10K pullups on SCL/SDA so unless you have your own pullups, have it match the microcontroller's logic level too!
- **V+** - This is an *optional* power pin that will supply distributed power to the servos. If you are not using for servos you can leave disconnected. It is not used at all by the chip. You can also inject power from the 2-pin terminal block at the top of the board. You should provide 5-6VDC if you are using servos. If you have to, you can go higher to 12VDC, but if you mess up and connect VCC to V+ you could damage your board!

Control Pins

- **SCL** - I2C clock pin, connect to your microcontrollers I2C clock line. Can use 3V or 5V logic, and has a weak pullup to **VCC**
- **SDA** - I2C data pin, connect to your microcontrollers I2C data line. Can use 3V or 5V logic, and has a weak pullup to **VCC**
- **OE** - Output enable. Can be used to quickly disable all outputs. When this pin is *low* all pins are *enabled*. When the pin is *high* the outputs are *disabled*. Pulled low by default so it's an optional pin!

Output Ports

There are 16 output ports. Each port has 3 pins: V+, GND and the PWM output. Each PWM runs completely independently *but* they must all have the same PWM frequency. That is, for LEDs you probably want 1.0 KHz but servos

need 60 Hz - so you cannot use half for LEDs @ 1.0 KHz and half @ 60 Hz.

They're set up for servos but you can use them for LEDs! Max current per pin is 25mA.

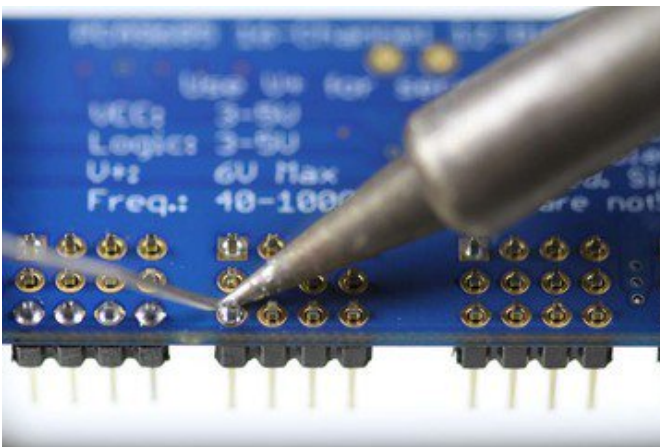
There are 220 ohm resistors in series with all PWM Pins and the output logic is the same as **VCC** so keep that in mind if using LEDs.

Assembly



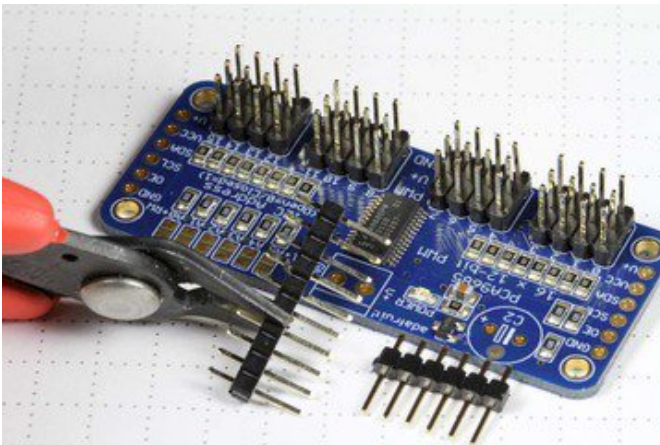
Install the Servo Headers

Install 4 3x4 pin male headers into the marked positions along the edge of the board.



Solder all pins

There are a lot of them!

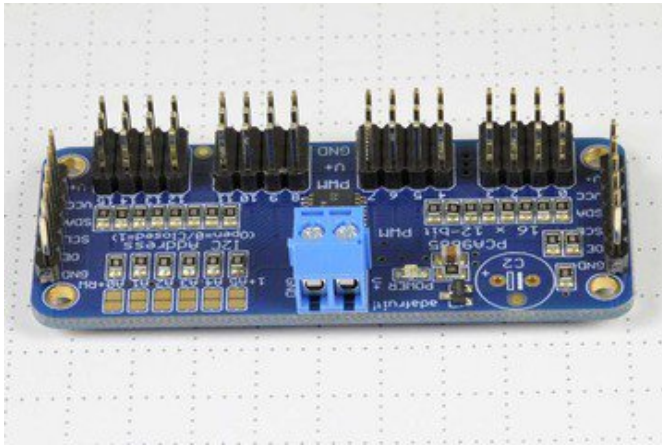


Add Headers for Control

A strip of male header is included. Where you want to install headers and on what side depends a little on use:

- For [breadboard use](http://adafruit.it/239) (<http://adafruit.it/239>), install headers on the bottom of the board.
- For use with [jumper wires](http://adafruit.it/758) (<http://adafruit.it/758>), install the headers on top of the board.
- For use with our [6-pin cable](http://adafruit.it/206) (<http://adafruit.it/206>), install the headers on top of the board.

If you are chaining multiple driver boards, you will want headers on both ends.



Install Power Terminals

If you are chaining multiple driver boards, you only need a power terminal on the first one.

Hooking it Up

Connecting to the Arduino

The PWM/Servo Driver uses I2C so it takes only 4 wires to connect to your Arduino:

"Classic" Arduino wiring:

- +5v -> VCC (this is power for the BREAKOUT only, NOT the servo power!)
- GND -> GND
- Analog 4 -> SDA
- Analog 5 -> SCL

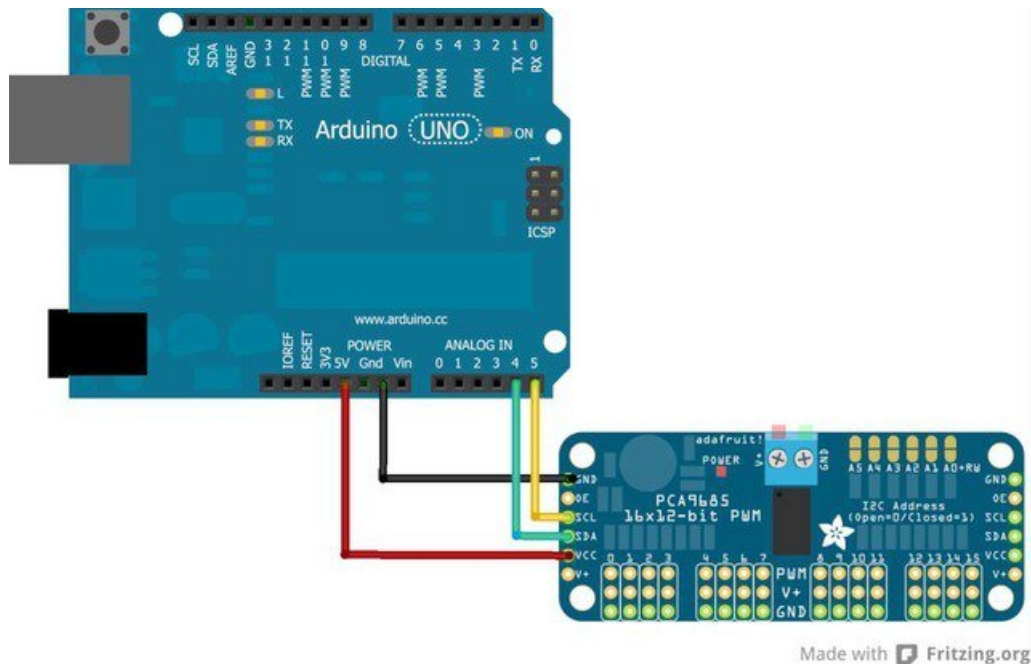
Older Mega wiring:

- +5v -> VCC (this is power for the BREAKOUT only, NOT the servo power!)
- GND -> GND
- Digital 20 -> SDA
- Digital 21 -> SCL

R3 and later Arduino wiring (Uno, Mega & Leonardo):

(These boards have dedicated SDA & SCL pins on the header nearest the USB connector)

- +5v -> VCC (this is power for the BREAKOUT only, NOT the servo power!)
- GND -> GND
- SDA -> SDA
- SCL -> SCL



The VCC pin is just power for the chip itself. If you want to connect servos or LEDs that use the V+ pins, you MUST connect the V+ pin as well. The V+ pin can be as high as 6V even if VCC is 3.3V (the chip is 5V safe). We suggest connecting power through the blue terminal block since it is polarity protected.

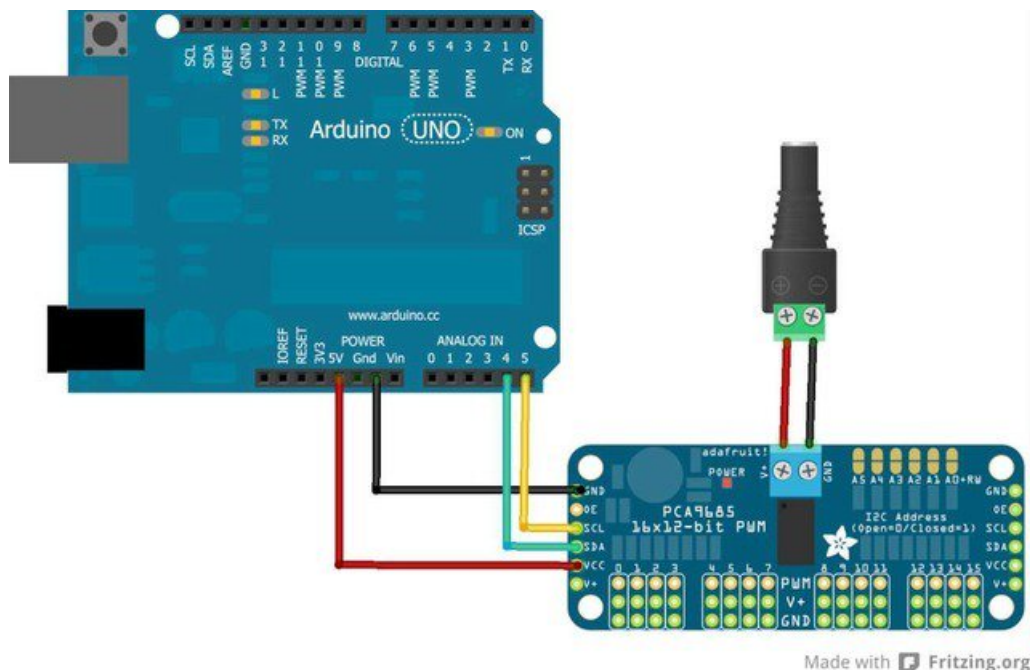
Power for the Servos

Most servos are designed to run on about 5 or 6v. Keep in mind that a lot of servos moving at the same time (particularly large powerful ones) will need a lot of current. Even micro servos will draw several hundred mA when moving. Some High-torque servos will draw more than 1A each under load.

Good power choices are:

- 5v 2A switching power supply
- 5v 10A switching power supply
- 4xAA Battery Holder - 6v with Alkaline cells. 4.8v with NiMH rechargeable cells.
- 4.8 or 6v Rechargeable RC battery packs from a hobby store.

It is not a good idea to use the Arduino 5v pin to power your servos. Electrical noise and 'brownouts' from excess current draw can cause your Arduino to act erratically, reset and/or overheat.

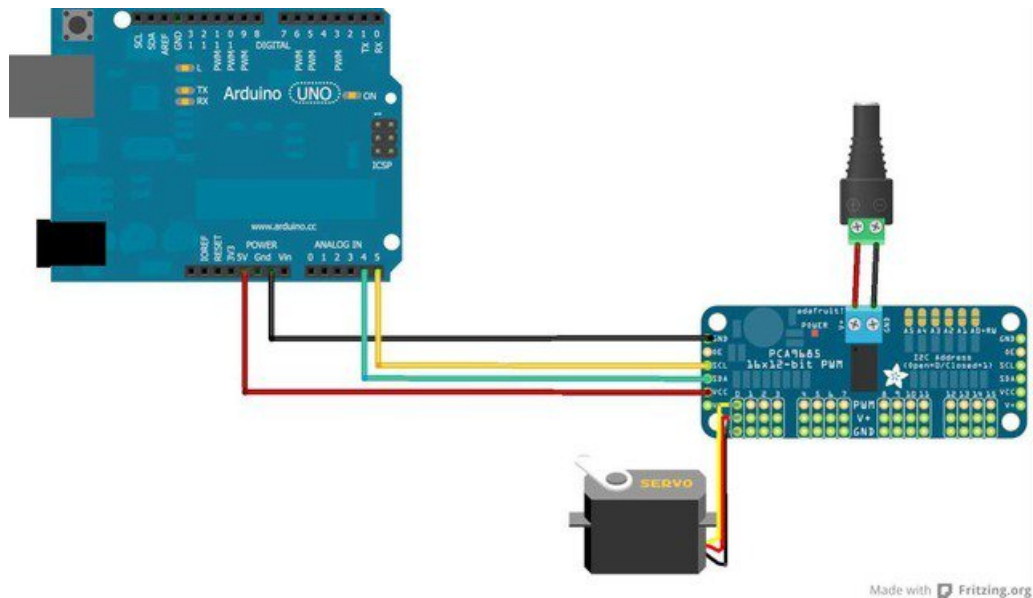


Adding a Capacitor to the thru-hole capacitor slot

We have a spot on the PCB for soldering in an electrolytic capacitor. Based on your usage, you may or may not need a capacitor. If you are driving a lot of servos from a power supply that dips a lot when the servos move, $n * 100\mu\text{F}$ where n is the number of servos is a good place to start - eg **470 μF** or more for 5 servos. Since its so dependent on servo current draw, the torque on each motor, and what power supply, there is no "one magic capacitor value" we can suggest which is why we don't include a capacitor in the kit.

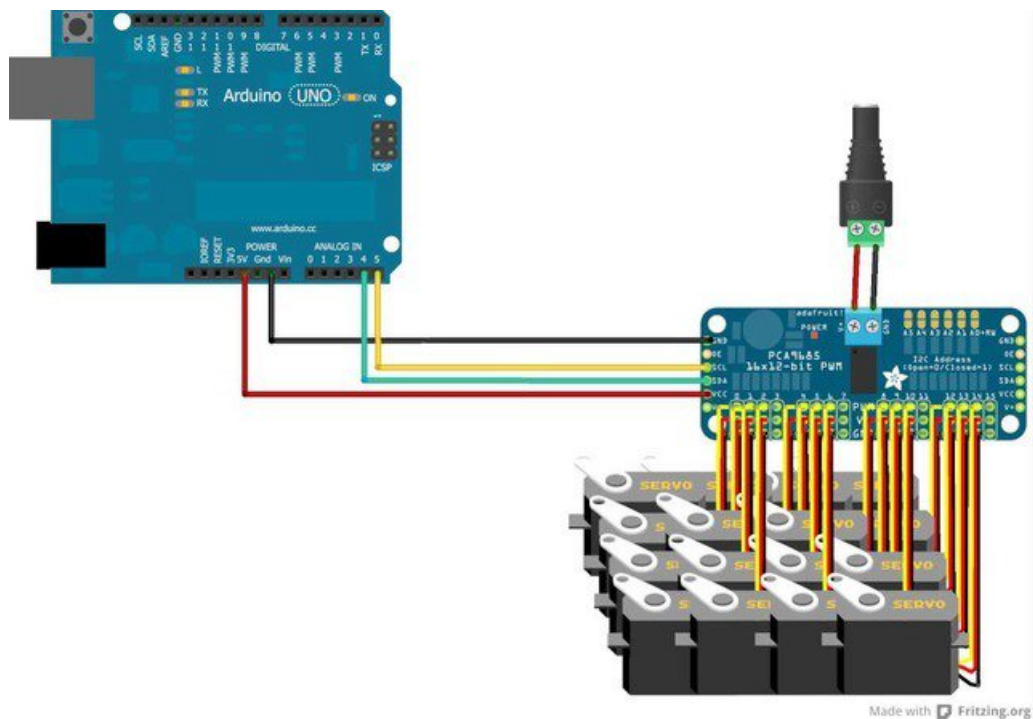
Connecting a Servo

Most servos come with a standard 3-pin female connector that will plug directly into the headers on the Servo Driver. Be sure to align the plug with the ground wire (usually black or brown) with the bottom row and the signal wire (usually yellow or white) on the top.



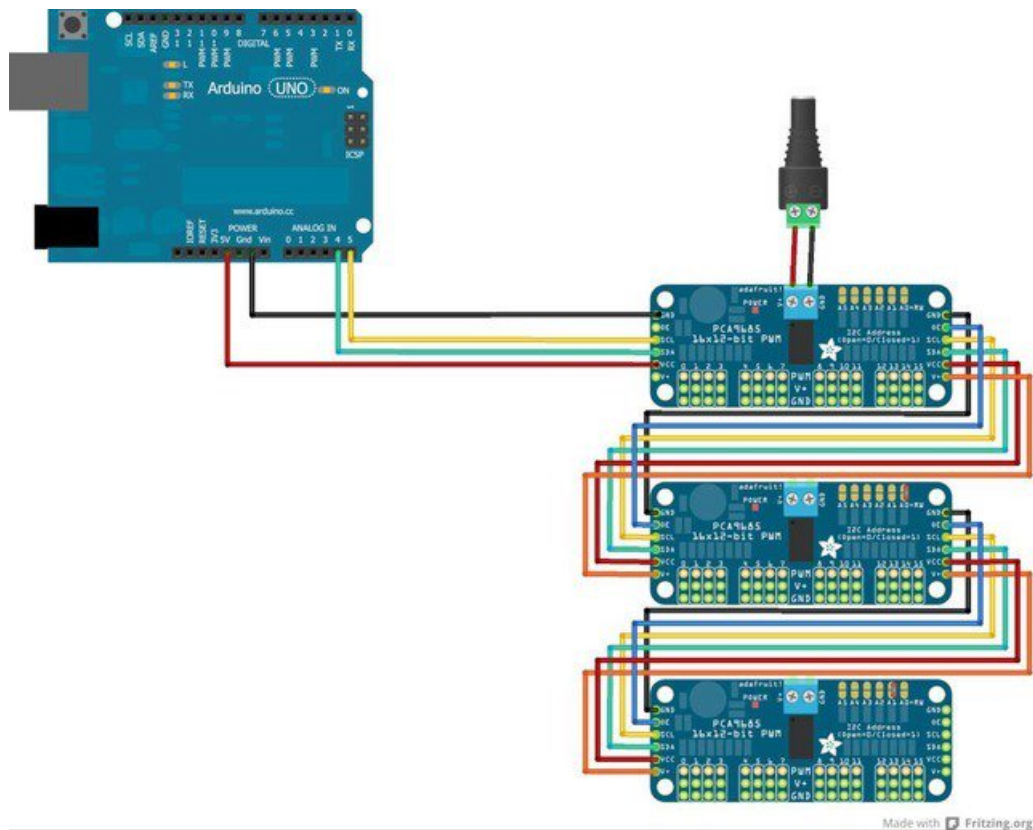
Adding More Servos

Up to 16 servos can be attached to one board. If you need to control more than 16 servos, additional boards can be chained as described on the next page.



Chaining Drivers

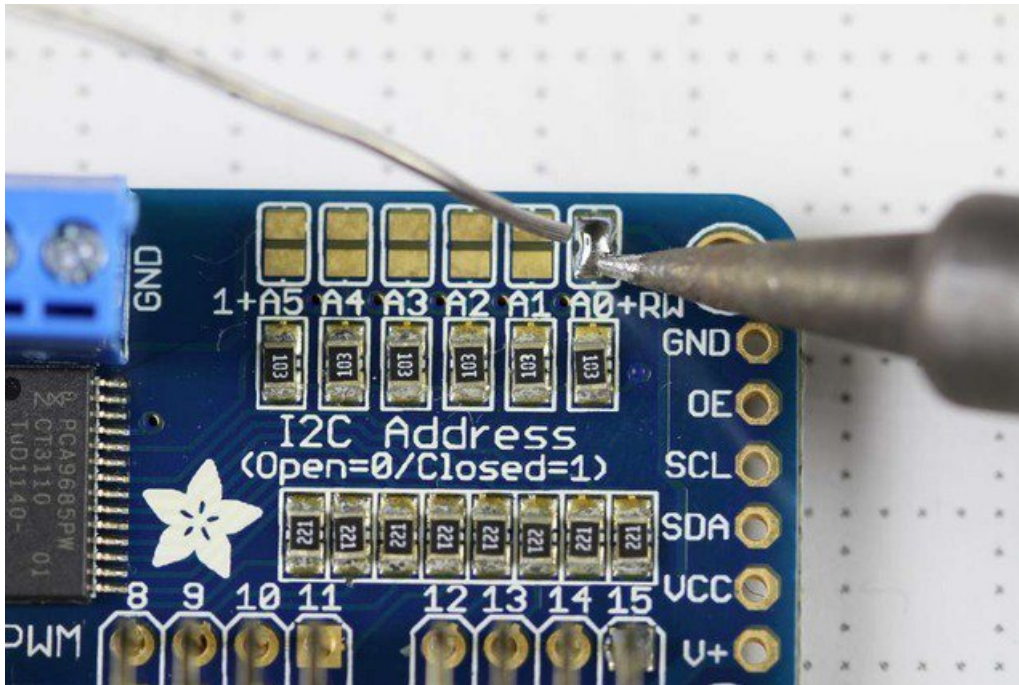
Multiple Drivers (up to 62) can be chained to control still more servos. With headers at both ends of the board, the wiring is as simple as connecting a 6-pin parallel cable from one board to the next.



Addressing the Boards

Each board in the chain must be assigned a unique address. This is done with the address jumpers on the upper right edge of the board. The I2C base address for each board is 0x40. The binary address that you program with the address jumpers is added to the base I2C address.

To program the address offset, use a drop of solder to bridge the corresponding address jumper for each binary '1' in the address.



Board 0: Address = 0x40 Offset = binary 00000 (no jumpers required)
 Board 1: Address = 0x41 Offset = binary 00001 (bridge A0 as in the photo above)
 Board 2: Address = 0x42 Offset = binary 00010 (bridge A1)
 Board 3: Address = 0x43 Offset = binary 00011 (bridge A0 & A1)
 Board 4: Address = 0x44 Offset = binary 00100 (bridge A2)

etc.

In your sketch, you'll need to declare a separate pobjct for each board. Call begin on each object, and control each servo through the object it's attached to. For example:

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

Adafruit_PWMServoDriver pwm1 = Adafruit_PWMServoDriver(0x40);
Adafruit_PWMServoDriver pwm2 = Adafruit_PWMServoDriver(0x41);

void setup() {
  Serial.begin(9600);
  Serial.println("16 channel PWM test!");

  pwm1.begin();
  pwm1.setPWMPFreq(1600); // This is the maximum PWM frequency

  pwm2.begin();
  pwm2.setPWMPFreq(1600); // This is the maximum PWM frequency
}
```

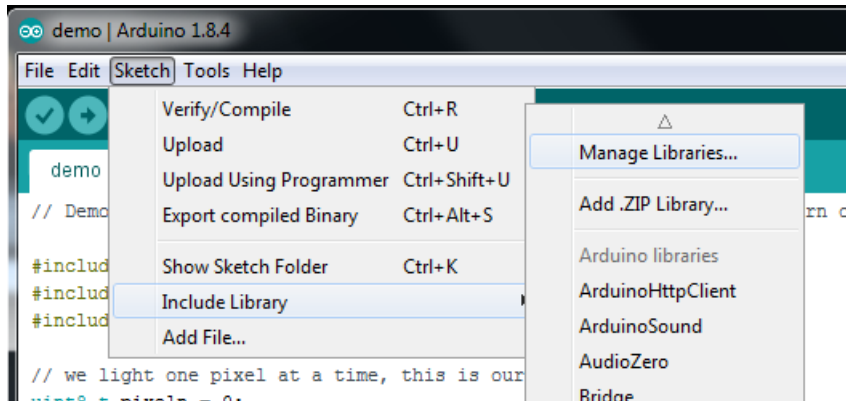

Using the Adafruit Library

Since the PWM Servo Driver is controlled over I2C, its super easy to use with any microcontroller or microcomputer. In this demo we'll show using it with the Arduino IDE but the C++ code can be ported easily

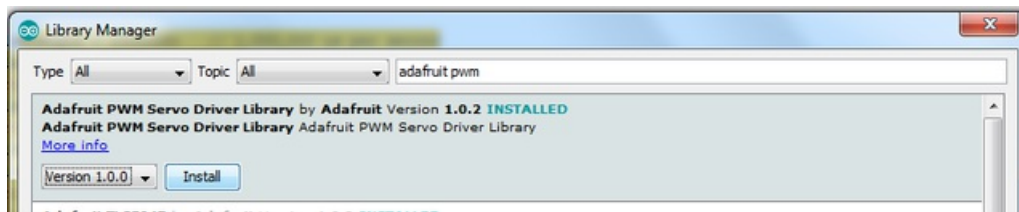
Install Adafruit PCA9685 library

To begin reading sensor data, you will need to [install the Adafruit_PWMServo library \(code on our github repository\)](#). It is available from the Arduino library manager so we recommend using that.

From the IDE open up the library manager...



And type in **adafruit pwm** to locate the library. Click **Install**

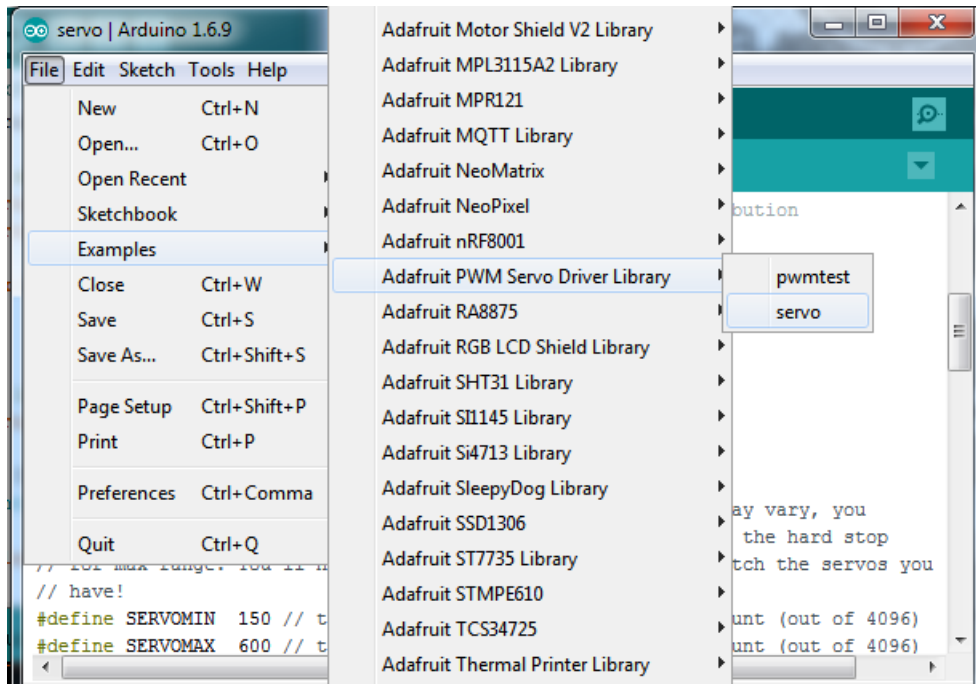


We also have a great tutorial on Arduino library installation at:
<http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use>

Test with the Example Code:

First make sure all copies of the Arduino IDE are closed.

Next open the Arduino IDE and select **File->Examples->Adafruit_PWMServoDriver->Servo**. This will open the example file in an IDE window.



If using a Breakout:

Connect the driver board and servo as shown on the previous page. Don't forget to provide power to both **Vin** (3-5V logic level) and **V+** (5V servo power). **Check the green LED is lit!**

If using a Shield:

Plug the shield into your Arduino. Don't forget you will also have to provide 5V to the V+ terminal block. **Both red and green LEDs must be lit.**

If using a FeatherWing:

Plug the FeatherWing into your Feather. Don't forget you will also have to provide 5V to the V+ terminal block. **Check the green LED is lit!**

Connect a Servo

A single servo should be plugged into the **PWM #0** port, the first port. You should see the servo sweep back and forth over approximately 180 degrees.

Calibrating your Servos

Servo pulse timing varies between different brands and models. Since it is an analog control circuit, there is often some variation between samples of the same brand and model. For precise position control, you will want to calibrate the minimum and maximum pulse-widths in your code to match known positions of the servo.

Find the Minimum:

Using the example code, edit SERVOMIN until the low-point of the sweep reaches the minimum range of travel. It is best to approach this gradually and stop before the physical limit of travel is reached.

Find the Maximum:

Again using the example code, edit SERVOMAX until the high-point of the sweep reaches the maximum range of travel. Again, is best to approach this gradually and stop before the physical limit of travel is reached.

Use caution when adjusting SERVOMIN and SERVOMAX. Hitting the physical limits of travel can strip the gears and permanently damage your servo.

Converting from Degrees to Pulse Length

The [Arduino "map\(\)" function](#) is an easy way to convert between degrees of rotation and your calibrated SERVOMIN and SERVOMAX pulse lengths. Assuming a typical servo with 180 degrees of rotation; once you have calibrated SERVOMIN to the 0-degree position and SERVOMAX to the 180 degree position, you can convert any angle between 0 and 180 degrees to the corresponding pulse length with the following line of code:

```
pulselength = map(degrees, 0, 180, SERVOMIN, SERVOMAX);
```

Library Reference

setPWMFreq(freq)

Description

This function can be used to adjust the PWM frequency, which determines how many full 'pulses' per second are generated by the IC. Stated differently, the frequency determines how 'long' each pulse is in duration from start to finish, taking into account both the high and low segments of the pulse.

Frequency is important in PWM, since setting the frequency too high with a very small duty cycle can cause problems, since the 'rise time' of the signal (the time it takes to go from 0V to VCC) may be longer than the time the signal is active, and the PWM output will appear smoothed out and may not even reach VCC, potentially causing a number of problems.

Arguments

- **freq**: A number representing the frequency in Hz, between 40 and 1000

Example

The following code will set the PWM frequency to the maximum value of 1000Hz:

```
pwm.setPWMFreq(1000)
```

setPWM(channel, on, off)

Description

This function sets the start (on) and end (off) of the high segment of the PWM pulse on a specific channel. You specify the 'tick' value between 0..4095 when the signal will turn on, and when it will turn off. Channel indicates which of the 16 PWM outputs should be updated with the new values.

Arguments

- **channel**: The channel that should be updated with the new values (0..15)
- **on**: The tick (between 0..4095) when the signal should transition from low to high
- **off**: the tick (between 0..4095) when the signal should transition from high to low

Example

The following example will cause channel 15 to start low, go high around 25% into the pulse (tick 1024 out of 4096), transition back to low 75% into the pulse (tick 3072), and remain low for the last 25% of the pulse:

```
pwm.setPWM(15, 1024, 3072)
```

Using as GPIO

There's also some special settings for turning the pins fully on or fully off

You can set the pin to be fully on with

```
pwm.setPWM(pin, 4096, 0);
```

You can set the pin to be fully off with

```
pwm.setPWM(pin, 0, 4096);
```


CircuitPython

This guide is for version 3.0.0 of the PCA9685 library. Make sure to use a bundle from 20180110 or later.

Adafruit CircuitPython Module Install

To use the PCA9685 with your [Adafruit CircuitPython](#) board you'll need to install the [Adafruit_CircuitPython_PCA9685](#) module on your board. **Remember this module is for Adafruit CircuitPython firmware and not MicroPython.org firmware!**

First make sure you are running the [latest version of Adafruit CircuitPython](#) for your board. Next you'll need to install the necessary libraries to use the hardware--read below and carefully follow the referenced steps to find and install these libraries from [Adafruit's CircuitPython library bundle](#) with a version 20180110 or newer.

Bundle Install

For express boards that have extra flash storage, like the Feather/Metro M0 express and Circuit Playground express, you can easily install the necessary libraries with [Adafruit's CircuitPython bundle](#). This is an all-in-one package that includes the necessary libraries to use the PCA9685 with CircuitPython. To install the bundle follow the steps in your board's guide, like [these steps for the Feather M0 express board](#).

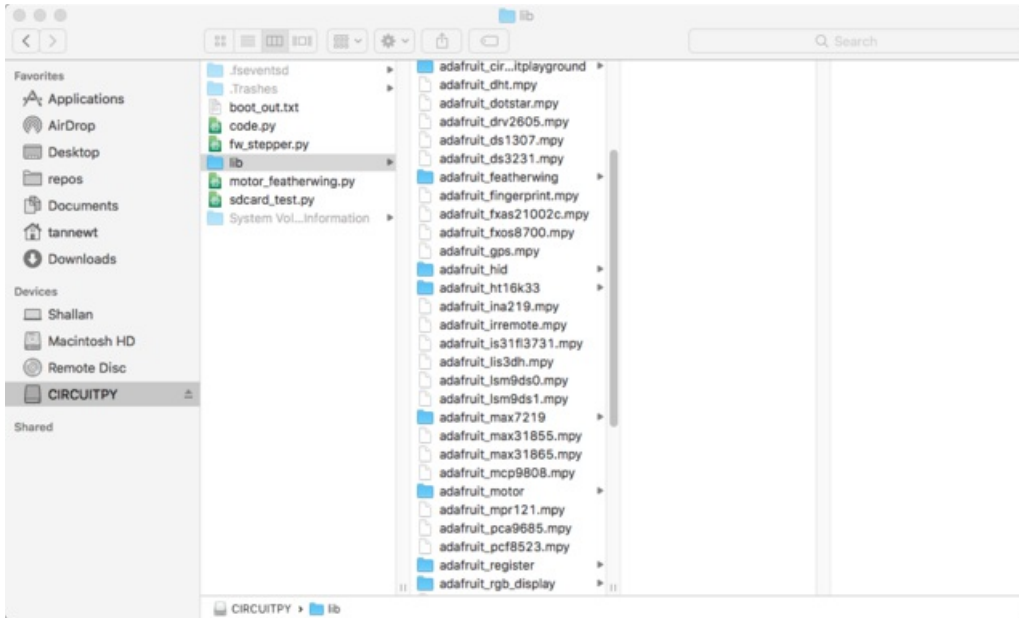
Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries from the bundle:

- `adafruit_pca9685`
- `adafruit_bus_device`
- `adafruit_register`
- `adafruit_motor`

If your board supports USB mass storage, like the M0-based boards, then simply drag the files to the board's file system. **Note on boards without external SPI flash, like a Feather M0 or Trinket/Gemma M0, you might run into issues on Mac OSX with hidden files taking up too much space when drag and drop copying, [see this page for a workaround](#).**

If your board doesn't support USB mass storage, like the ESP8266, then [use a tool like ampy to copy the file to the board](#). You can use the latest version of ampy and its [new directory copy command](#) to easily move module directories to the board.

Before continuing make sure your board's lib folder or root filesystem has the `adafruit_pca9685`, `adafruit_bus_device`, `adafruit_register` and `adafruit_motor` folders/modules copied over.



Usage

The following section will show how to control the PCA9685 from the board's Python prompt / REPL. You'll learn how to interactively control servos and dim LEDs by typing in the code below.

First [connect to the board's serial REPL](#) so you are at the CircuitPython >>> prompt.

I2C Initialization

First you'll need to initialize the I2C bus for your board. First import the necessary modules:

```
import board
import busio
```

Now for either board run this command to create the I2C instance using the default SCL and SDA pins (which will be marked on the boards pins if using a Feather or similar Adafruit board):

```
i2c = busio.I2C(board.SCL, board.SDA)
```

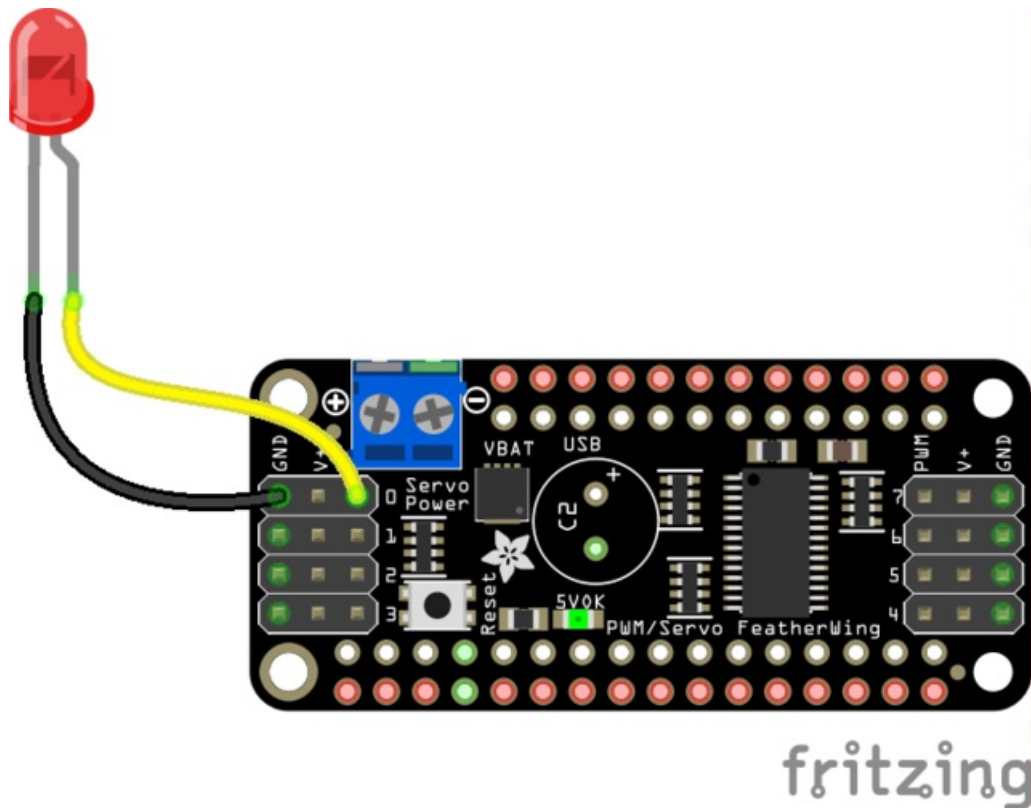
After initializing the I2C interface you need to import the PCA9685 module to use it in your own code:

```
import adafruit_pca9685
```

Dimming LED's

Each channel of the PCA9685 can be used to control the brightness of an LED. The PCA9685 generates a high-speed PWM signal which turns the LED on and off very quickly. If the LED is turned on longer than turned off it will appear brighter to your eyes.

First wire a LED to the board as follows. Note you don't need to use a resistor to limit current through the LED as the PCA9685 will limit the current to around 10mA:



Fritzing Source

<https://adafru.it/zew>

- LED cathode / shorter leg to PCA9685 channel GND / ground.
- LED anode / longer leg to PCA9685 channel PWM.

Now in the Python REPL you can create an instance of the basic PCA9685 class which provides low-level PWM control of the board's channels:

```
pca = adafruit_pca9685.PCA9685(i2c)
```

The PCA9685 class provides control of the PWM frequency and each channel's duty cycle. Check out the [PCA9685 class documentation](#) for more details.

For dimming LEDs you typically don't need to use a fast PWM signal frequency and can set the board's PWM frequency to 60hz by setting the **frequency** attribute:

```
pca.frequency = 60
```

The PCA9685 supports 16 separate channels that share a frequency but can have independent duty cycles. That way you could dim 16 LEDs separately!

The PCA9685 object has a **channels** attribute which has an object for each channel that can control the duty cycle. To get the individual channel use the `[]` to index into **channels**.

```
led_channel = pca.channels[0]
```

Now control the LED brightness by controlling the duty cycle of the channel connected to the LED. The duty cycle value should be a 16-bit value, i.e. 0 to 0xffff, which represents what percent of the time the signal is on vs. off. A value of 0xffff is 100% brightness, 0 is 0% brightness, and in-between values go from 0% to 100% brightness.

For example set the LED completely on with a duty cycle of 0xffff:

```
led_channel.duty_cycle = 0xffff
```

After running the command above you should see the LED light up at full brightness!

Now turn the LED off with a duty cycle of 0:

```
led_channel.duty_cycle = 0
```

Try an in-between value like 1000:

```
led_channel.duty_cycle = 1000
```

You should see the LED dimly lit. Try experimenting with other duty cycle values to see how the LED changes brightness!

For example make the LED glow on and off by setting **duty_cycle** in a loop:

```
# Increase brightness:
for i in range(0xffff):
    led_channel.duty_cycle = i

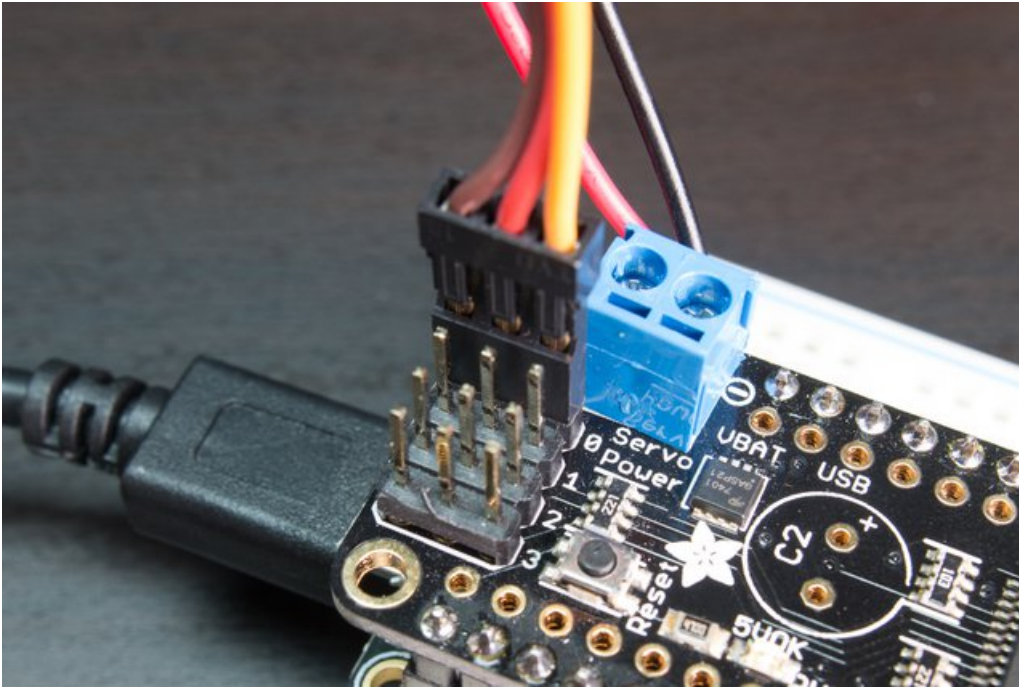
# Decrease brightness:
for i in range(0xffff, 0, -1):
    led_channel.duty_cycle = i
```

These for loops take a while because 16-bits is a lot of numbers. **CTRL-C** to stop the loop from running and return to the REPL.

Control Servos

Servo motors use a PWM signal to control the position of their arm or horn. Using the PCA9685 and the [Motor library](#) you can easily plug in servos and control them with Python. If you aren't familiar with servos be sure to first read this [intro to servos page](#) and this [in-depth servo guide page](#).

First connect the servo to a channel on the PCA9685. Be sure to plug the servo connector in the correct way! Check your servo's datasheet to be sure, but typically the brown wire is connected to ground, the red wire is connected to 5V power, and the yellow pin is connected to PWM:



Be sure you've turned on or plugged in the external 5V power supply to the PCA9685 board too!

Now in the Python REPL as above with the led, save a variable for its channel:

```
servo_channel = pca.channels[0]
```

Servos typically operate at a frequency of 50 hz so update pca accordingly.

```
pca.frequency = 50
```

Now that the PCA9685 is set up for servos lets make a Servo object so that we can adjust the servo based on **angle** instead of **duty_cycle**.

By default the Servo class will use actuation range, minimum pulse-width, and maximum pulse-width values that should work for most servos. However [check the Servo class documentation](#) for more details on extra parameters to customize the signal generated for your servos.

```
import adafruit_motor.servo
servo = adafruit_motor.servo.Servo(servo_channel)
```

With Servo, you specify a position as an angle. The angle will always be between 0 and the actuation range given when Servo was created. The default is 180 degrees but your servo might have a smaller sweep--change the total angle by specifying the **actuation_angle** parameter in the Servo class initializer above.

Now set the angle to 180, one extreme of the range:

```
servo.angle = 180
```

Or to sweep back to the minimum 0 degree position:

```
servo.angle = 0
```

Often the range an individual servo recognizes varies a bit from other servos. If the servo didn't sweep the full expected range, then try adjusting `min_pulse` and `max_pulse`. Lower `min_pulse` until the servo stops moving or moves irregularly when angle is changed to 0. Raise `max_pulse` until the servo stops moving or moves irregularly when angle is change to the actuation range.

```
servo = adafruit_motor.servo.Servo(servo_channel, min_pulse=800, max_pulse=2200)
```

That's all there is to controlling servos with the PCA9685 and CircuitPython! Using the **angle** attribute you can sweep and move servos in any way. This is perfect for building robots, actuating switches, or other fun mechanical projects!

Files

Schematic & Fabrication Print

C.4 Hitec HS422 Servo



Hitec HS422 Servo

SKU:SER0002



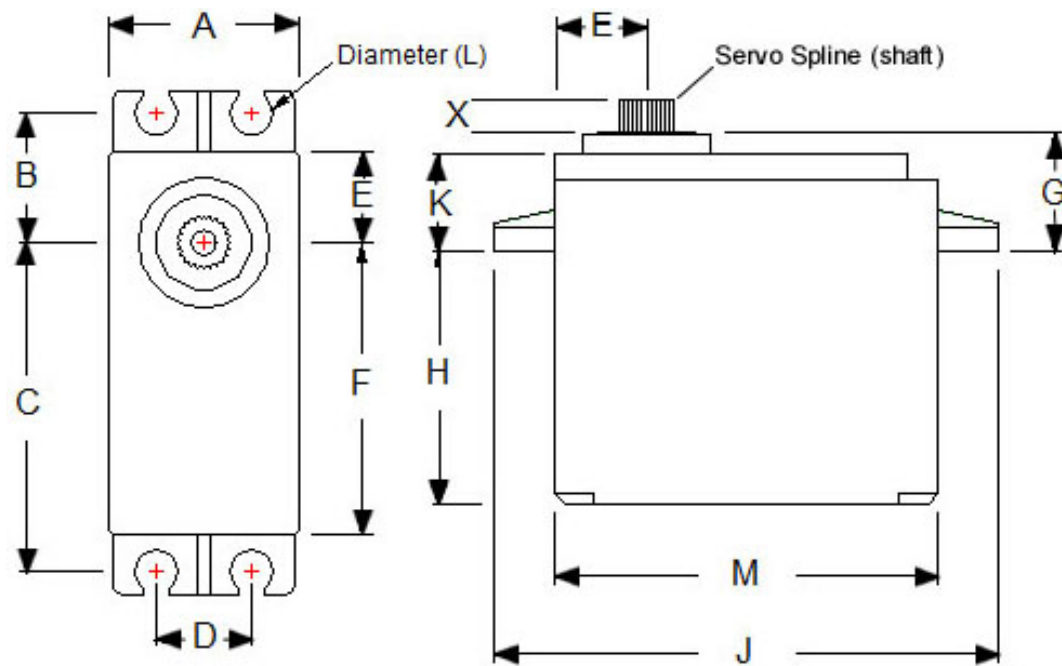
INTRODUCTION

The HS-422 has been around for a while and for a good reason, durability and the ability to be easily modified. For inexpensive robotic applications this the perfect servo. It can easily be modified for wheeled applications since the output shaft is supported on the bottom and top with bronze bushings and the potentiometer is indirect drive. The circuit board is also separate from the motor so the potentiometer can be easily accessed. Simply a great servo at a great price.

SPECIFICATION

- Control System: +Pulse Width Control 1500usec Neutral
- Required Pulse: 3-5 Volt Peak to Peak Square Wave
- Operating Voltage: 4.8-6.0 Volts
- Operating Temperature Range: -20 to +60 Degree C
- Operating Speed (4.8V): 0.21sec/60° at no load
- Operating Speed (6.0V): 0.16sec/60° at no load
- Stall Torque (4.8V): 45.82 oz/in. (3.3kg.cm)
- Stall Torque (6.0V): 56.93 oz/in. (4.1kg.cm)
- Operating Angle: 45 Deg. one side pulse traveling 400usec
- 360 Modifiable: Yes
- Direction: Clockwise/Pulse Traveling 1500 to 1900usec
- Current Drain (4.8V): 8mA/idle and 150mA no load operating
- Current Drain (6.0V): 8.8mA/idle and 180mA no load operating
- Dead Band Width: 8usec
- Motor Type: 3 Pole Ferrite
- Potentiometer Drive: Indirect Drive
- Bearing Type: Dual Oilite Bushing
- Gear Type: Nylon
- Connector Wire Length: 11.81" (300mm=11.81in)
- Dimensions: See Schematics
- Weight: 1.6oz (45.5g)





A = .780" (19.82mm=0.78in)
B = .530" (13.47mm=0.53in)
C = 1.33" (33.79mm=1.33in)
D = .400" (10.17mm=0.4in)
E = .380" (9.66mm=0.38in)
F = 1.19" (30.22mm=1.19in)
G = .460" (11.68mm=0.46in)
H = 1.05" (26.67mm=1.05in)
J = 2.08" (52.84mm=2.08in)
K = .368" (9.35mm=0.37in)
L = .172" (4.38mm=0.17in)
M = 1.57" (39.88mm=1.57in)
X = .120" (3.05mm=0.12in)

SHIPPING LIST

- Hitec HS422 Servo x1

C.5 Microsoft HoloLens



Microsoft HoloLens

Microsoft HoloLens is the first fully self-contained holographic computer running Windows 10. It is completely untethered—no wires, phones, or connection to a PC needed. Microsoft HoloLens allows you to place holograms in your physical environment providing a heads-up, hands-free way to see your world.

HoloLens Device Specifications

Software	Windows 10 Windows Mixed Reality	Wireless	Wi-Fi 802.11ac wireless networking Bluetooth 4.1 Low Energy (LE) wireless connectivity
Weight	579g (1.28 lbs.)	Audio	3D audio speakers 3.5mm audio jack
Optics / Display	2.3 megapixel widescreen see-through holographic lenses (waveguides) 2 HD 16:9 light engines (screen aspect ratio) Holographic Density: >2.5k radiant (light points per radian) 1 2.4-megapixel photographic video camera Automatic pupillary distance calibration	Ports	Micro USB 2.0
Sensors	1 IMU (Accelerometer, gyroscope, and magnetometer) 4 environment sensors 1 energy-efficient depth camera with a 120°x120° angle of view Four-microphone array 1 ambient light sensor	Physical Buttons	Power Volume up/down Brightness up/down
Processors	Intel 32-bit (1GHz) with TPM 2.0 support Custom-built Microsoft Holographic Processing Unit (HPU 1.0)	What's in the box	HoloLens Development Edition Clicker Carrying case Charger and cable Microfiber cloth Nose pads Overhead strap
Memory	2GB RAM	OS and Apps	Windows 10 Calibration Holograms Learn Gestures Settings Windows Feedback Windows Store Microsoft Edge Photos
Storage	64GB (flash memory)	Hardware / Software Requirements	Windows 10 PC Visual Studio 2015 Unity
Power	Battery Life 2-3 hours of active use Up to 2 weeks on standby mode Fully functional when charging Passively cooled (no fans) Battery status LED nodes (battery level and power/standby mode settings)		
Security	Windows 10 software updates Additional security and device management available for Commercial Suite		

Human understanding capabilities

The holograms you'll see with Microsoft HoloLens can appear lifelike, and can move, be shaped, and change according to interaction with you or the physical environment in which they are visible. Interact with holograms using the navigation commands below:

Spatial sound	Allows the user to hear binaural audio which can simulate spatial effects, meaning the user, virtually can perceive, and locate a sound, as though it is coming from a virtual pinpoint or location. Learn more: https://www.youtube.com/watch?v=aB3TDjYklmo
Gaze tracking	Allows the user to bring application focus to whatever the user is perceiving to navigate and explore, the technology can tell exactly what and where to show the images for each pupil to generate stereoscopic 3D illusions. Learn more: https://www.youtube.com/watch?v=zCPiZlWdVws
Gesture input	Allows the user to use the "bloom" gesture to pull up a UI navigation menu screen (similar to a Windows key on a Windows keyboard, this is your "home" button) Use the air tap gesture to select menu commands (similar to clicking an imaginary computer mouse) Learn more: https://www.youtube.com/watch?v=kwn9Lh0E_vU
Voice support	Allows the user to use voice commands (similar to asking Cortana, Siri, Google a question) Allows developers to use the Text to Speech capability (i.e. speech recognition) to create voice inputs for apps they are creating in Unity. Learn more: https://www.youtube.com/watch?v=eHMKOpNUtR8

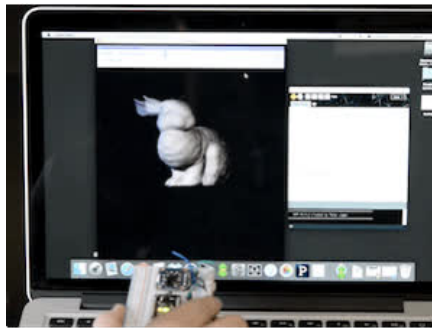
	Commercial Suite	Development Edition
HoloLens device and clicker	Included	Included
App dev support: forums/community	Included	Included
1 year OEM warranty	Included	-
Enterprise features		-
Kiosk mode	Included	-
Mobile Device Management	Included	-
Identity with PIN unlock	Included	-
Windows Update for Business	Included	-
Data security	Included	-
Work/remote access	Included	-
Windows Store for Business	Included	-
Devices available	US/Canada, Australia, France, Germany, Ireland, Puerto Rico, New Zealand, United Kingdom	US/Canada, Australia, France, Germany, Ireland, Puerto Rico, New Zealand, United Kingdom
Supported languages	US - English only	US - English only
Purchase at Microsoft Store (hololens.com)	Online	Online
Sales channel	A minimum of 5 units; Order handled by a Microsoft account representative	-

C.6 Adafruit BNO055 Absolute Orientation Sensor



Adafruit BNO055 Absolute Orientation Sensor

Created by Kevin Townsend



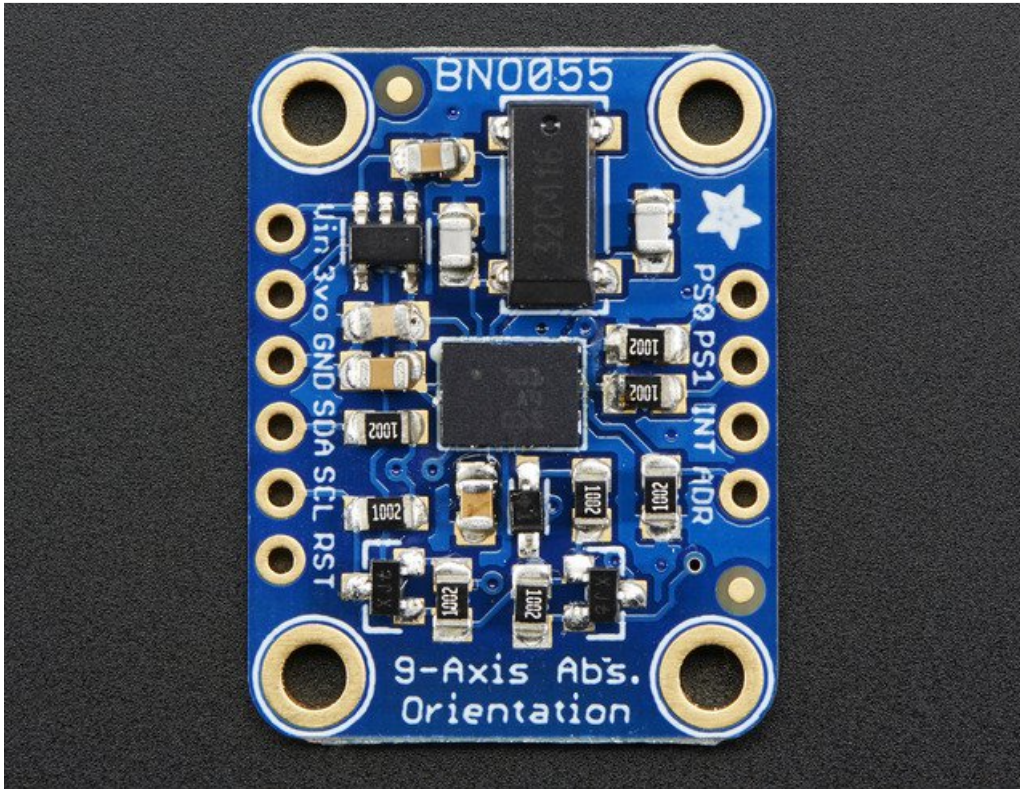
Last updated on 2018-04-29 11:32:36 PM UTC

Guide Contents

Guide Contents	2
Overview	4
Data Output	4
Related Resources	5
Pinouts	6
Power Pins	6
I2C Pins	6
Other Pins	6
Assembly	7
Prepare the header strip:	7
Add the breakout board:	8
And Solder!	8
Arduino Code	11
Wiring for Arduino	11
Software	11
Download the Driver from Github	11
Download Adafruit_Sensor	12
Adafruit Unified Sensor System	12
'sensorapi' Example	13
Raw Sensor Data	14
.getVector (adafruit_vector_type_t vector_type)	14
.getQuat(void)	15
.getTemp(void)	15
'rawdata' Example	15
Processing Test	17
Requirements	17
Opening the Processing Sketch	17
Run the Bunny Sketch on the Uno	18
Rabbit Disco!	18
Device Calibration	21
Interpreting Data	21
Generating Calibration Data	22
Persisting Calibration Data	22
Bosch Video	22
CircuitPython Code	23
Usage	24
FAQs	26
Can I manually set the calibration constants?	26
Does the device make any assumptions about its initial orientation?	26
Why doesn't Euler output seem to match the Quaternion output?	26
I'm sometimes losing data over I2C, what can I do about this?	26
I have some high frequency (> 2MHz) wires running near the BNO055 and I'm getting unusual results/hanging behavior	27

Downloads	28
Files	28
Pre-Compiled Bunny Rotate Binaries	28
Schematic	28
Board Dimensions	28

Overview



If you've ever ordered and wire up a 9-DOF sensor, chances are you've also realized the challenge of turning the sensor data from an accelerometer, gyroscope and magnetometer into actual "3D space orientation"! Orientation is a hard problem to solve. The sensor fusion algorithms (the secret sauce that blends accelerometer, magnetometer and gyroscope data into stable three-axis orientation output) can be mind-numbingly difficult to get right and implement on low cost real time systems.

Bosch is the first company to get this right by taking a MEMS accelerometer, magnetometer and gyroscope and putting them on a single die with a high speed ARM Cortex-M0 based processor to digest all the sensor data, abstract the sensor fusion and real time requirements away, and spit out data you can use in quaternions, Euler angles or vectors.

Rather than spending weeks or months fiddling with algorithms of varying accuracy and complexity, you can have meaningful sensor data in minutes thanks to the BNO055 - a smart 9-DOF sensor that does the sensor fusion all on its own!

Data Output

The BNO055 can output the following sensor data:

- **Absolute Orientation** (Euler Vector, 100Hz)
Three axis orientation data based on a 360° sphere
- **Absolute Orientation** (Quaternion, 100Hz)
Four point quaternion output for more accurate data manipulation
- **Angular Velocity Vector** (100Hz)
Three axis of 'rotation speed' in rad/s
- **Acceleration Vector** (100Hz)

Three axis of acceleration (gravity + linear motion) in m/s^2

- **Magnetic Field Strength Vector** (20Hz)

Three axis of magnetic field sensing in micro Tesla (μT)

- **Linear Acceleration Vector** (100Hz)

Three axis of linear acceleration data (acceleration minus gravity) in m/s^2

- **Gravity Vector** (100Hz)

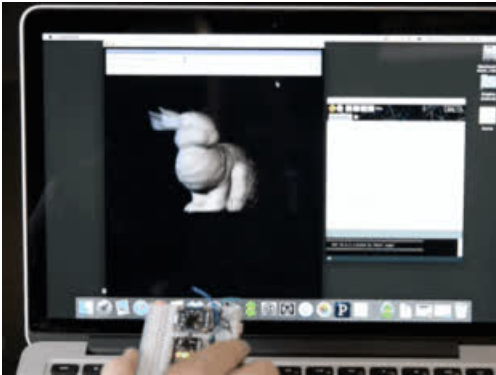
Three axis of gravitational acceleration (minus any movement) in m/s^2

- **Temperature** (1Hz)

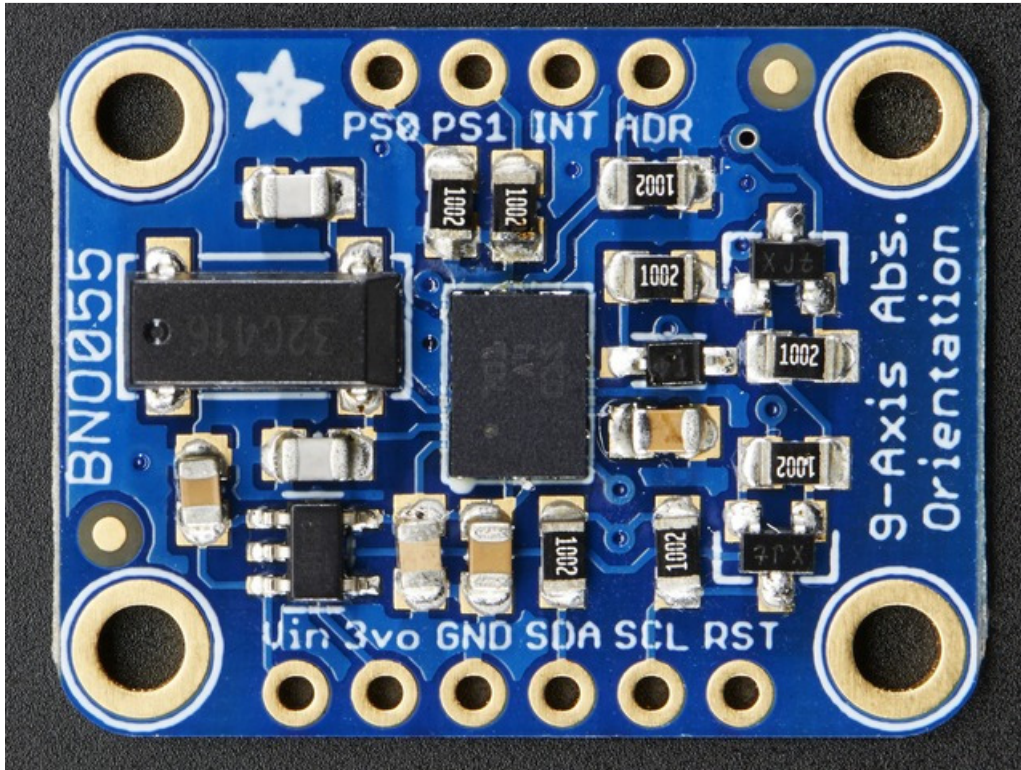
Ambient temperature in degrees celsius

Related Resources

- [Datasheet](#)
- [Adafruit BNO055 Library](#) (Github)



Pinouts



Power Pins

- **VIN:** 3.3-5.0V power supply input
- **3VO:** 3.3V output from the on-board linear voltage regulator, you can grab up to about 50mA as necessary
- **GND:** The common/GND pin for power and logic

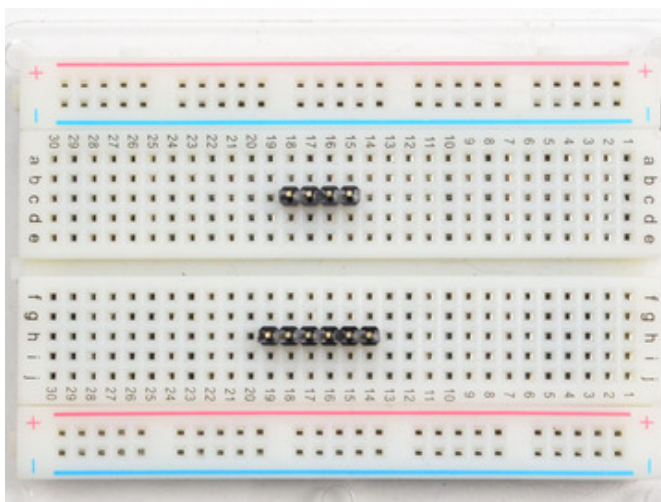
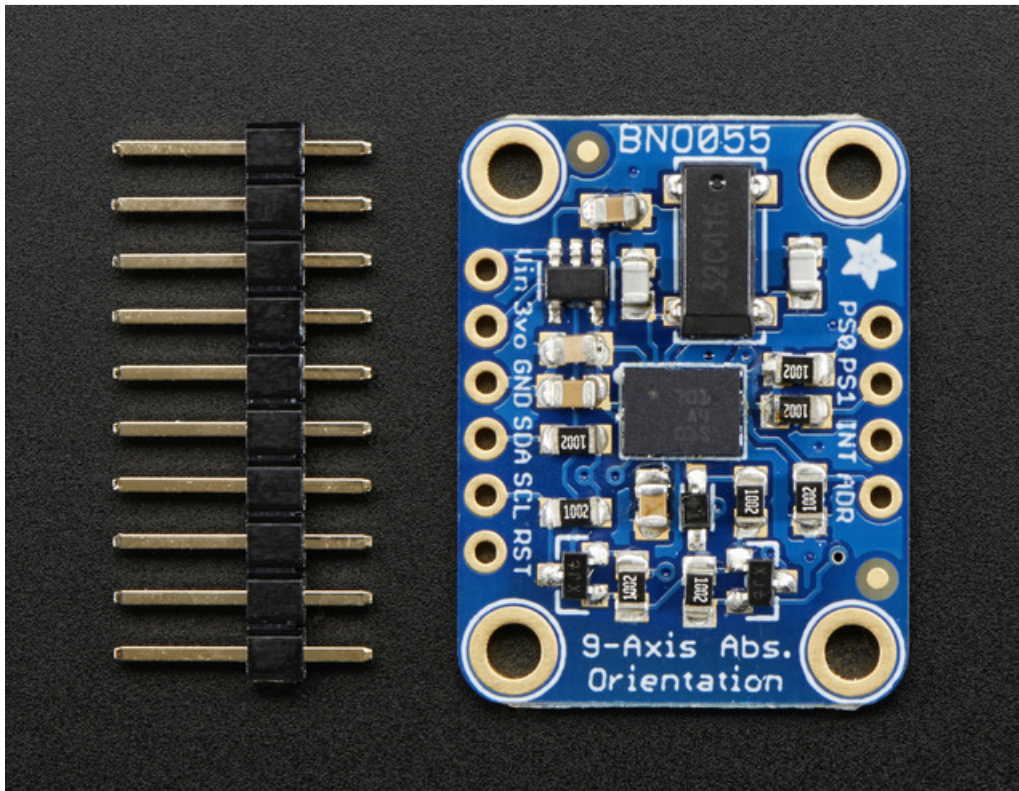
I2C Pins

- **SCL** - I2C clock pin, connect to your microcontrollers I2C clock line. This pin can be used with 3V or 5V logic, and there's a 10K pullup on this pin.
- **SDA** - I2C data pin, connect to your microcontrollers I2C data line. This pin can be used with 3V or 5V logic, and there's a 10K pullup on this pin.

Other Pins

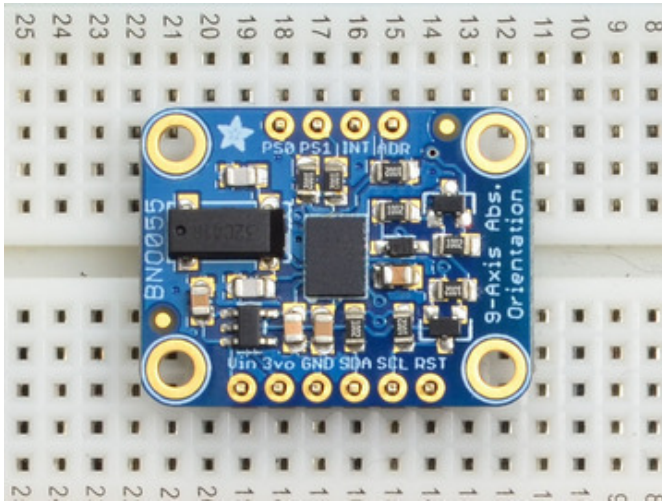
- **RST:** Hardware reset pin. Set this pin low then high to cause a reset on the sensor. This pin is 5V safe.
- **INT:** The HW interrupt output pin, which can be configured to generate an interrupt signal when certain events occur like movement detected by the accelerometer, etc. (not currently supported in the Adafruit library, but the chip and HW is capable of generating this signal). The voltage level out is 3V
- **ADR:** Set this pin high to change the default I2C address for the BNO055 if you need to connect two ICs on the same I2C bus. The default address is 0x28. If this pin is connected to 3V, the address will be 0x29
- **PS0 and PS1:** These pins can be used to change the mode of the device (it can also do HID-I2C and UART) and also are provided in case Bosch provides a firmware update at some point for the ARM Cortex M0 MCU inside the sensor. They should normally be left unconnected.

Assembly



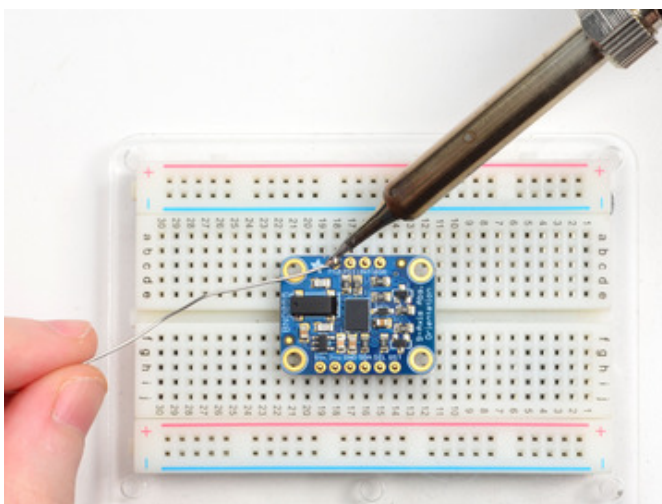
Prepare the header strip:

Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**



Add the breakout board:

Place the breakout board over the pins so that the short pins poke through the breakout pads

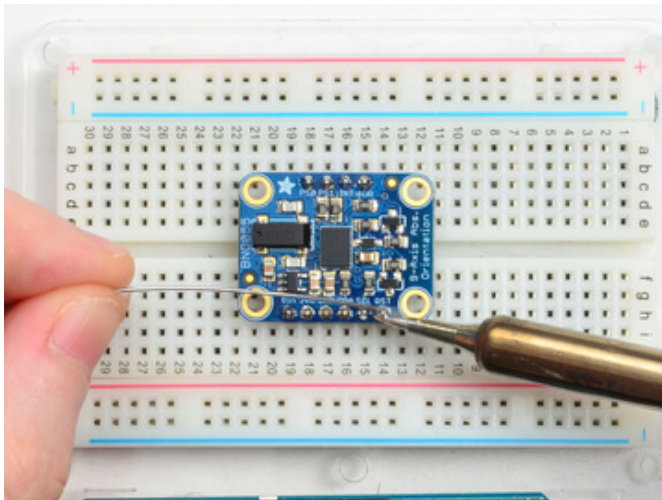
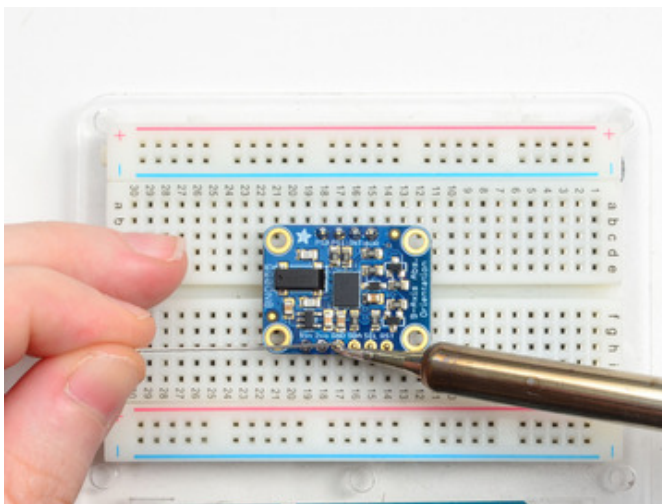
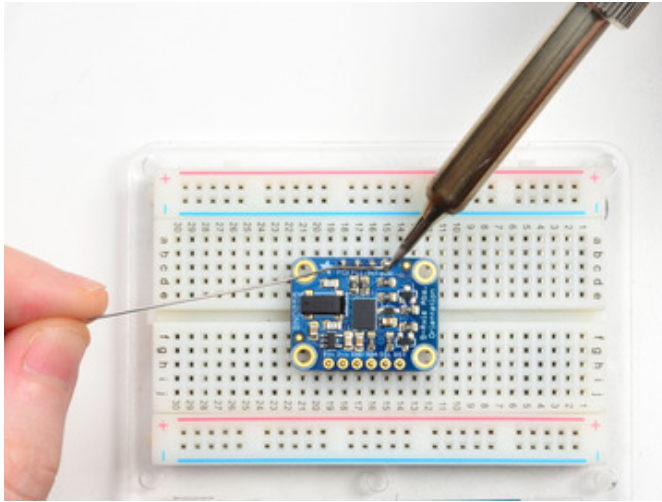


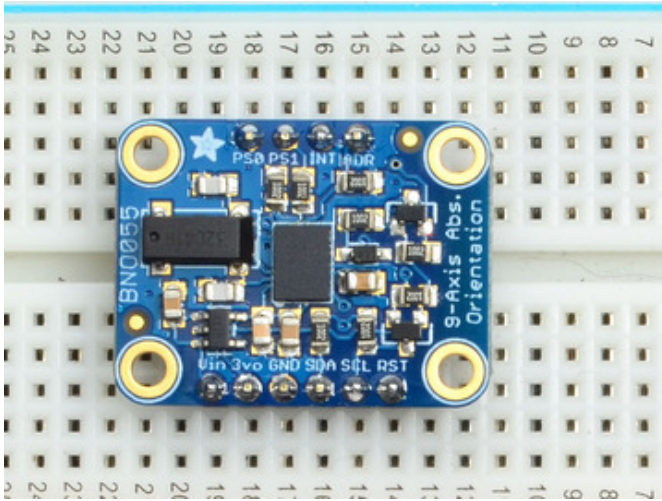
And Solder!

Be sure to solder all pins for reliable electrical contact.

Solder the longer power/data strip first

(For tips on soldering, be sure to check out our [Guide to Excellent Soldering](https://adafruit.it/aTk) (<https://adafruit.it/aTk>)).





You're done! Check your solder joints visually and continue onto the next steps

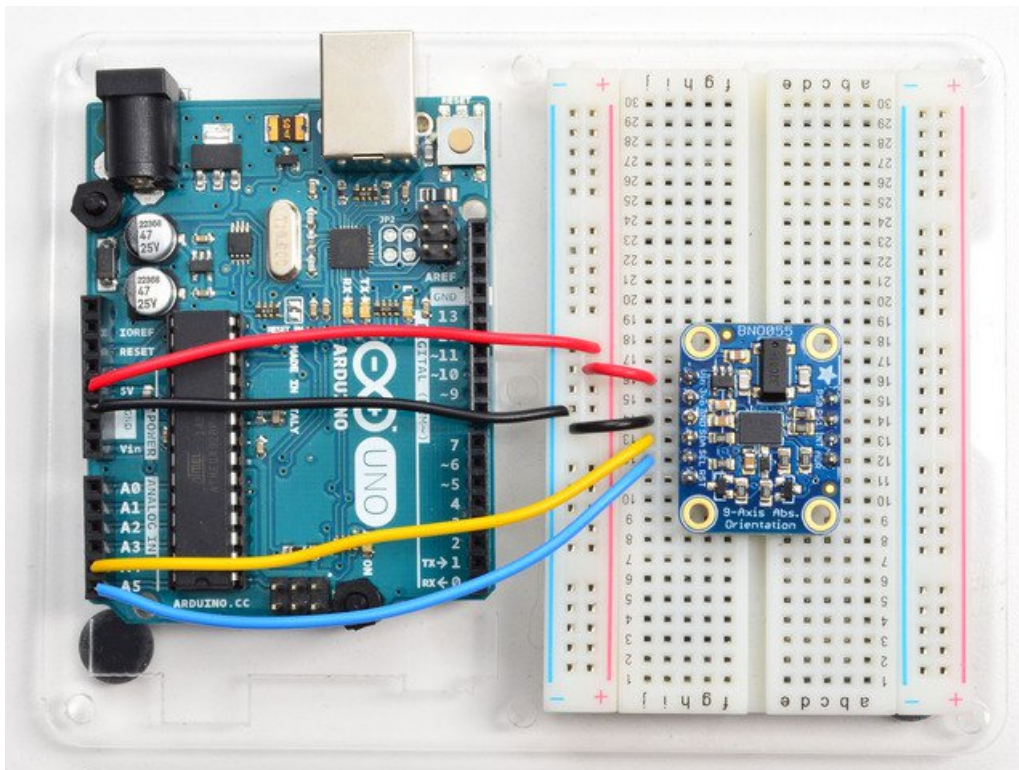
Arduino Code

Wiring for Arduino

You can easily wire this breakout to any microcontroller, we'll be using an Arduino. For another kind of microcontroller, just make sure it has I2C capability, then port the code - its pretty simple stuff!

To connect the assembled BNO055 breakout to an Arduino Uno, follow the wiring diagram below.

- Connect **Vin** to the power supply, 3-5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V
- Connect **GND** to common power/data ground
- Connect the **SCL** pin to the I2C clock **SCL** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A5**, on a Mega it is also known as **digital 21** and on a Leonardo/Micro, **digital 3**
- Connect the **SDA** pin to the I2C data **SDA** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A4**, on a Mega it is also known as **digital 20** and on a Leonardo/Micro, **digital 2**



If you're using a Genuino Zero or Arduino Zero with the built in EDBG interface you may need to use I2C address 0x29 since 0x28 is 'taken' by the DBG chip

Software

The [Adafruit_BNO055 driver](#) supports reading raw sensor data, or you can use the [Adafruit Unified Sensor](#) system to retrieve orientation data in a standard data format.

Download the Driver from Github

To begin controlling the motor chip, you will need to [download the Adafruit_BNO055 Library from our github](#)

[repository](#). You can do that by visiting the github repo and manually downloading or, easier, just click this button to download the zip

Download Adafruit_BNO055 from Github

<https://adafru.it/f0K>

Rename the uncompressed folder **Adafruit_BNO055** and check that the **Adafruit_BNO055** folder contains **Adafruit_BNO055.cpp** and **Adafruit_BNO055.h**

Place the **Adafruit_BNO055** library folder your *arduinasketchfolder/libraries/* folder. You may need to create the **libraries** subfolder if its your first library. Restart the IDE.

We also have a great tutorial on Arduino library installation at:
<http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use>

Download Adafruit_Sensor

We also have a core sensor library that helps manage sensor readings. So, just like the BNO055 library, download [Adafruit_Sensor](#)

Install just like you did with **Adafruit_BNO055**

Download Adafruit_Sensor

<https://adafru.it/cMO>

Adafruit Unified Sensor System

Since the Adafruit_BNO055 driver is based on the Adafruit Unified Sensor system, you can retrieve your three axis orientation data (in Euler angles) using the standard types and functions described in the [Adafruit Sensor learning guide](#) ([.getEvent](#), [.getSensor](#), etc.).

This is probably the easiest option if all you care about is absolute orientation data across three axis.

For example, the following code snippet shows the core of what is needed to start reading data using the Unified Sensor System:

```

#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imuMaths.h>

Adafruit_BNO055 bno = Adafruit_BNO055(55);

void setup(void)
{
  Serial.begin(9600);
  Serial.println("Orientation Sensor Test"); Serial.println("");

  /* Initialise the sensor */
  if(!bno.begin())
  {
    /* There was a problem detecting the BNO055 ... check your connections */
    Serial.print("Ooops, no BNO055 detected ... Check your wiring or I2C ADDR!");
    while(1);
  }

  delay(1000);

  bno.setExtCrystalUse(true);
}

void loop(void)
{
  /* Get a new sensor event */
  sensors_event_t event;
  bno.getEvent(&event);

  /* Display the floating point data */
  Serial.print("X: ");
  Serial.print(event.orientation.x, 4);
  Serial.print("\tY: ");
  Serial.print(event.orientation.y, 4);
  Serial.print("\tZ: ");
  Serial.print(event.orientation.z, 4);
  Serial.println("");

  delay(100);
}

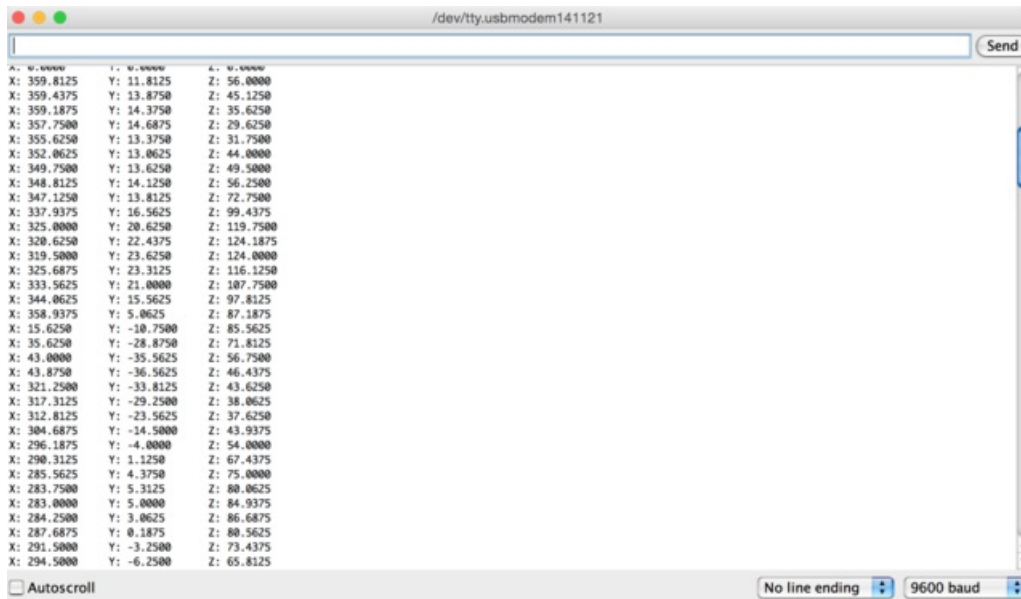
```

'sensorapi' Example

To test the Unified Sensor System output, open the **sensorapi** demo in the Adafruit_BNO055 examples folder:



This should produce the following output on the Serial Monitor:



Raw Sensor Data

If you don't want to use the Adafruit Unified Sensor system (for example if you want to access the raw accelerometer, magnetometer or gyroscope data directly before the sensor fusion algorithms process it), you can use the raw helper functions in the driver.

The key raw data functions are:

- `getVector` (adafruit_vector_type_t vector_type)
- `getQuat` (void)
- `getTemp` (void)

`.getVector (adafruit_vector_type_t vector_type)`

The `.getVector` function accepts a single parameter (`vector_type`), which indicates what type of 3-axis vector data to return.

The `vector_type` field can be one of the following values:

- **VECTOR_MAGNETOMETER** (values in uT, micro Teslas)
- **VECTOR_GYROSCOPE** (values in rps, radians per second)
- **VECTOR_EULER** (values in Euler angles or 'degrees', from 0..359)
- **VECTOR_ACCELEROMETER** (values in m/s^2)
- **VECTOR_LINEARACCEL** (values in m/s^2)
- **VECTOR_GRAVITY** (values in m/s^2)

For example, to get the Euler angles vector, we could run the following code:


```
imu::Vector<3> euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);

/* Display the floating point data */
Serial.print("X: ");
Serial.print(euler.x());
Serial.print(" Y: ");
Serial.print(euler.y());
Serial.print(" Z: ");
Serial.print(euler.z());
Serial.println("");
```

.getQuat(void)

The .getQuat function returns a Quaternion, which is often easier and more accurate to work with than Euler angles when doing sensor fusion or data manipulation with raw sensor data.

You can get a quaternion data sample via the following code:

```
imu::Quaternion quat = bno.getQuat();

/* Display the quat data */
Serial.print("qW: ");
Serial.print(quat.w(), 4);
Serial.print(" qX: ");
Serial.print(quat.x(), 4);
Serial.print(" qY: ");
Serial.print(quat.y(), 4);
Serial.print(" qZ: ");
Serial.print(quat.z(), 4);
Serial.println("");
```

.getTemp(void)

The .getTemp helper returns the current ambient temperature in degrees celsius, and can be read via the following function call:

```
/* Display the current temperature */
int8_t temp = bno.getTemp();

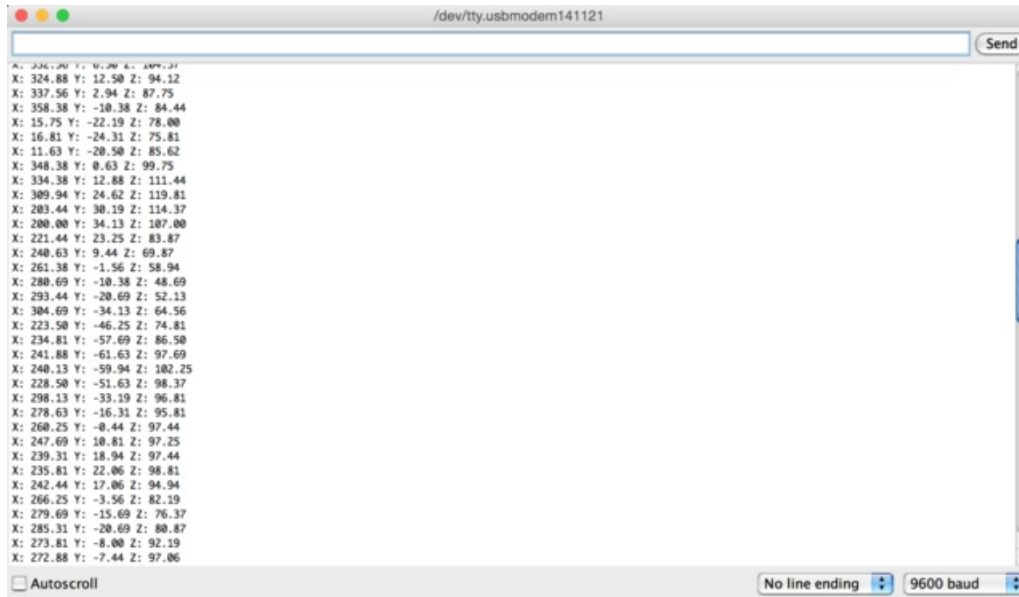
Serial.print("Current Temperature: ");
Serial.print(temp);
Serial.println(" C");
Serial.println("");
```

'rawdata' Example

To test the raw data output, open the **rawdata** demo in the Adafruit_BNO055 examples folder:



This should produce the following output on the Serial Monitor:



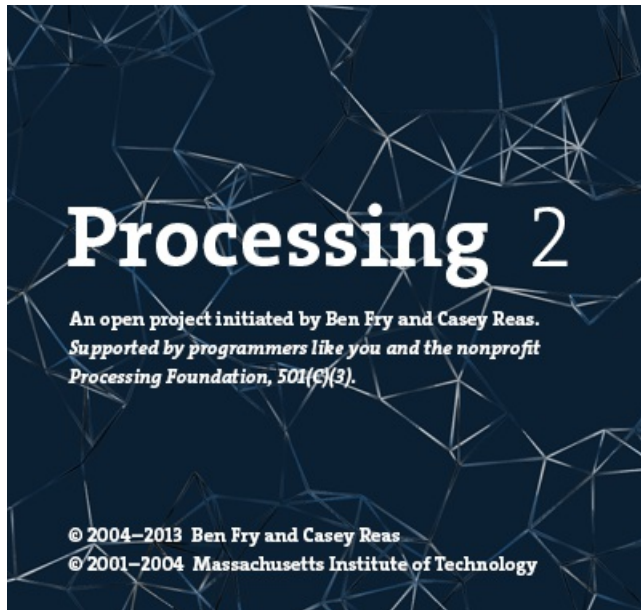
By default, the sketch generates **Euler angle** absolute orientation data, but you can easily modify the data displayed by changing the value provided to `.getVector` below:

```
// Possible vector values can be:
// - VECTOR_ACCELEROMETER - m/s^2
// - VECTOR_MAGNETOMETER  - uT
// - VECTOR_GYROSCOPE     - rad/s
// - VECTOR_EULER          - degrees
// - VECTOR_LINEARACCEL    - m/s^2
// - VECTOR_GRAVITY        - m/s^2
imu::Vector<3> euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);

/* Display the floating point data */
Serial.print("X: ");
Serial.print(euler.x());
Serial.print(" Y: ");
Serial.print(euler.y());
Serial.print(" Z: ");
Serial.print(euler.z());
Serial.println("");
```

Processing Test

To help you visualize the data, we've put together a basic Processing sketch that loads a 3D model (in the .obj file format) and renders it using the data generated by the BNO055 sketch on the Uno. The "bunny" sketch on the uno published data over UART, which the Processing sketch reads in, rotating the 3D model based on the incoming orientation data.



Requirements

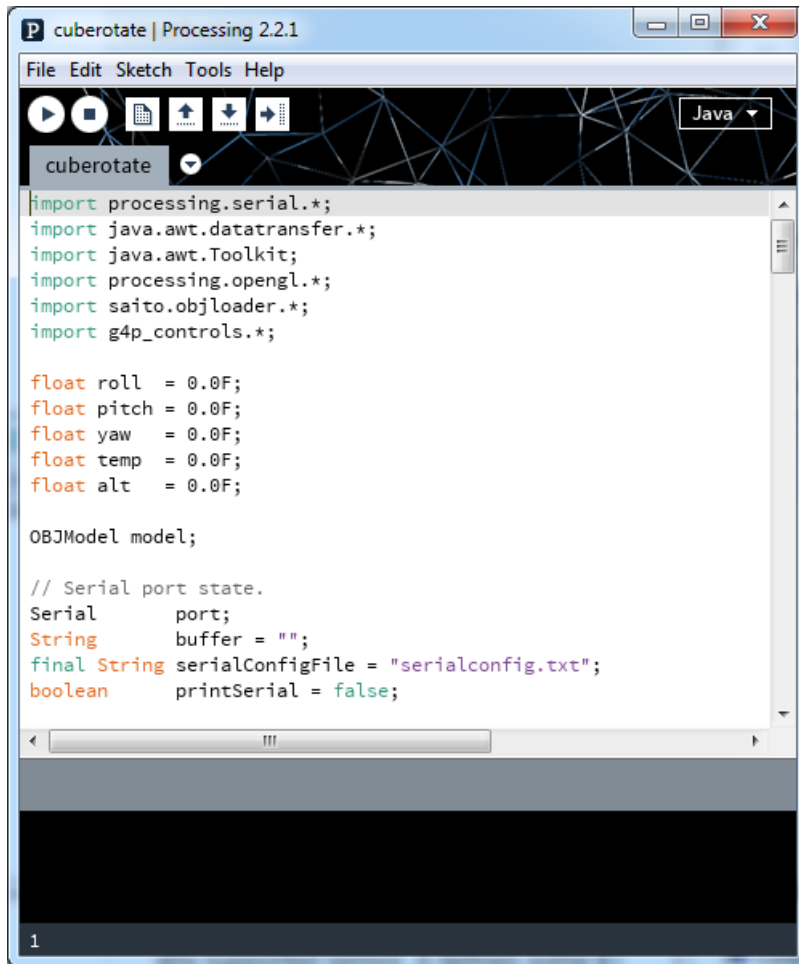
- [Processing 2.x](#)
 - Note that you can try later Processing versions like 3.0+ too. On some platforms Processing 2.2.1 has issues with supporting 3D acceleration (you might see 'NoClassDefFoundError: processing/awt/PGraphicsJava2D' errors). In those cases grab the later Processing 3.0+ release and use it instead of 2.x.
- [Saito's OBJ Loader](#) library for Processing (included as part of the Adafruit repo since Google Code is now 'End of Life').
- [G4P GUI library](#) for Processing ([download the latest version here](#) and copy the zip into the processing libraries folder along with the OBJ loader library above). Version 3.5.2 was used in this guide.

The OBJ library is required to load 3D models. It isn't strictly necessary and you could also render a boring cube in Processing, but why play with cubes when you have rabbits?!

Opening the Processing Sketch

The processing sketch to render the 3D model is contained in the sample folder as the ahrs sketch for the Uno.

With Processing open, navigate to you Adafruit_BNO055 library folder (ex.: **'libraries/Adafruit_BNO055'**), and open **'examples/bunny/processing/cuberotate/cuberotate.pde'**. You should see something like this in Processing:



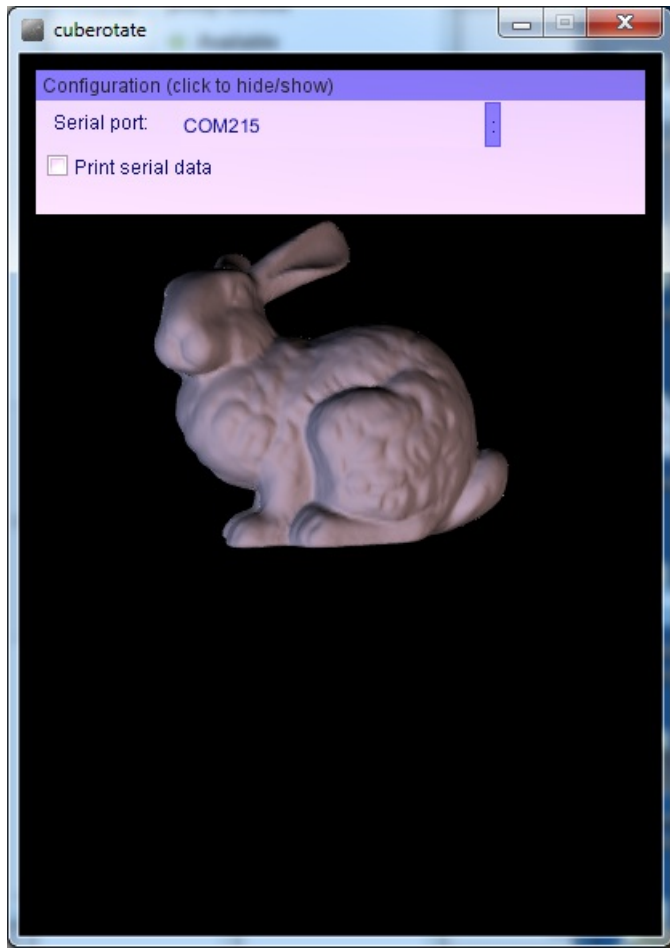
Run the Bunny Sketch on the Uno

Make sure that the "bunny" example sketch is running on the Uno, and that the Serial Monitor is **closed**.

With the sample sketch running on the Uno, click the triangular 'play' icon in Processing to start the sketch.

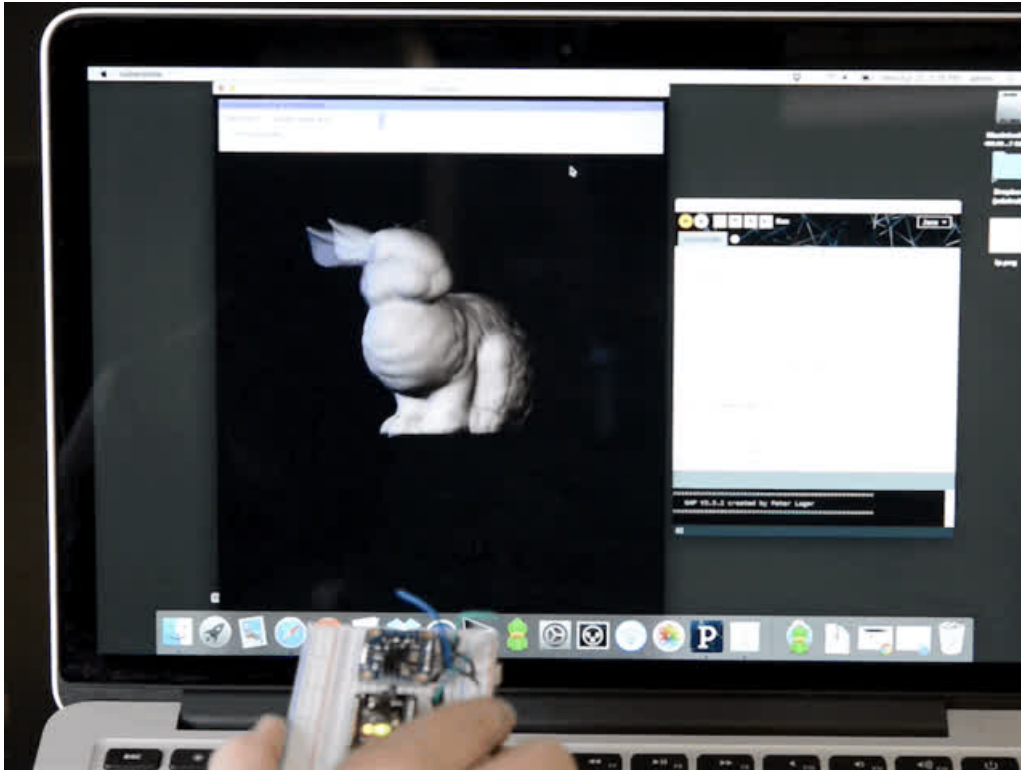
Rabbit Disco!

You should see a rabbit similar to the following image:



Before the rabbit will rotate you will need to click the : to the right of the serial port name. This will open a list of available serial ports, and you will need to click the appropriate serial port that your Arduino uses (check the Arduino IDE to see the port name if you're unsure). The chosen serial port should be remembered if you later run the sketch again.

As you rotate your breakout board, the rabbit should rotate to reflect the movement of the breakout in 3D-space, as seen in the video below



Also notice in the upper right corner of the dialog box at the top that the calibration of each sensor is displayed. It's important to calibrate the BNO055 sensor so that the most accurate readings are retrieved. Each sensor on the board has a separate calibration status from 0 (uncalibrated) up to 3 (fully calibrated). Check out the [video and information from this guide for how to best calibrate the BNO055 sensor](#).

Device Calibration

The BNO055 includes internal algorithms to constantly calibrate the gyroscope, accelerometer and magnetometer inside the device.

The exact nature of the calibration process is a black box and not fully documented, but you can read the calibration status of each sensor using the `.getCalibration` function in the [Adafruit_BNO055](#) library. An example showing how to use this function can be found in the sensorapi demo, though the code is also shown below for convenience sake.

The four calibration registers -- an overall system calibration status, as well individual gyroscope, magnetometer and accelerometer values -- will return a value between '0' (uncalibrated data) and '3' (fully calibrated). The higher the number the better the data will be.

```
/*
*****
*/
/*
  Display sensor calibration status
*/
*****
*/
void displayCalStatus(void)
{
  /* Get the four calibration values (0..3) */
  /* Any sensor data reporting 0 should be ignored, */
  /* 3 means 'fully calibrated' */
  uint8_t system, gyro, accel, mag;
  system = gyro = accel = mag = 0;
  bno.getCalibration(&system, &gyro, &accel, &mag);

  /* The data should be ignored until the system calibration is > 0 */
  Serial.print("\t");
  if (!system)
  {
    Serial.print("! ");
  }

  /* Display the individual values */
  Serial.print("Sys:");
  Serial.print(system, DEC);
  Serial.print(" G:");
  Serial.print(gyro, DEC);
  Serial.print(" A:");
  Serial.print(accel, DEC);
  Serial.print(" M:");
  Serial.println(mag, DEC);
}
```

Interpreting Data

The BNO055 will start supplying sensor data as soon as it is powered on. The sensors are factory trimmed to reasonably tight offsets, meaning you can get valid data even before the calibration process is complete, but particularly in NDOF mode **you should discard data as long as the system calibration status is 0 if you have the choice**.

The reason is that system cal '0' in NDOF mode means that the device has not yet found the 'north pole', and orientation values will be off. The heading will jump to an absolute value once the BNO finds magnetic north (the system calibration status jumps to 1 or higher).

When running in NDOF mode, any data where the system calibration value is '0' should generally be ignored

Generating Calibration Data

To generate valid calibration data, the following criteria should be met:

- **Gyroscope:** The device must be standing still in any position
- **Magnetometer:** In the past 'figure 8' motions were required in 3 dimensions, but with recent devices fast magnetic compensation takes place with sufficient normal movement of the device
- **Accelerometer:** The BNO055 must be placed in 6 standing positions for +X, -X, +Y, -Y, +Z and -Z. This is the most onerous sensor to calibrate, but the best solution to generate the calibration data is to find a block of wood or similar object, and place the sensor on each of the 6 'faces' of the block, which will help to maintain sensor alignment during the calibration process. You should still be able to get reasonable quality data from the BNO055, however, even if the accelerometer isn't entirely or perfectly calibrated.

Persisting Calibration Data

Once the device is calibrated, the calibration data will be kept until the BNO is powered off.

The BNO doesn't contain any internal EEPROM, though, so you will need to perform a new calibration every time the device starts up, or manually restore previous calibration values yourself.

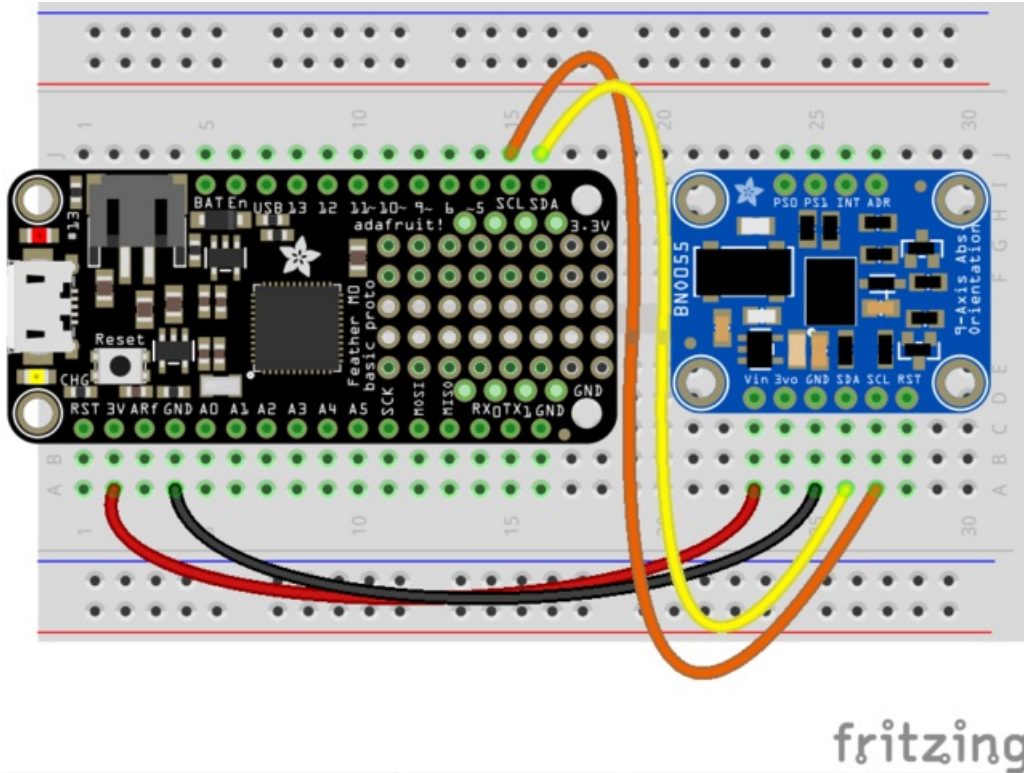
Bosch Video

Here's a video from the BNO055 makers on calibration!

CircuitPython Code

It's easy to use the BNO055 sensor with CircuitPython and the [Adafruit CircuitPython BNO055](#) library. This library allows you to easily write Python code that reads the acceleration and orientation of the sensor.

First wire up a BNO055 to your board exactly as shown on the previous pages for Arduino using the I2C interface. Here's an example of wiring a Feather M0 to the sensor with I2C:



- Board 3V to sensor VIN
- Board GND to sensor GND
- Board SCL to sensor SCL
- Board SDA to sensor SDA

Next you'll need to install the [Adafruit CircuitPython BNO055](#) library on your CircuitPython board. **Remember this module is for Adafruit CircuitPython firmware and not MicroPython.org firmware!**

First make sure you are running the [latest version of Adafruit CircuitPython](#) for your board.

Next you'll need to install the necessary libraries to use the hardware—carefully follow the steps to find and install these libraries from [Adafruit's CircuitPython library bundle](#). For example the Circuit Playground Express guide has [a great page on how to install the library bundle](#) for both express and non-express boards.

Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries from the bundle:

- `adafruit_bno055.mpy`
- `adafruit_bus_device`
- `adafruit_register`

You can also download the `adafruit_bno055.mpy` from [its releases page on Github](#).

Before continuing make sure your board's lib folder or root filesystem has the `adafruit_bno055.mpy`, `adafruit_bus_device`, and `adafruit_register` files and folders copied over.

Next [connect to the board's serial REPL](#) so you are at the CircuitPython `>>>` prompt.

Usage

To demonstrate the usage of the sensor we'll initialize it and read the acceleration, orientation (in Euler angles), and more from the board's Python REPL. First initialize the I2C bus and create an instance of the sensor by running:

```
import board
import busio
import adafruit_bno055
i2c = busio.I2C(board.SCL, board.SDA)
sensor = adafruit_bno055.BNO055(i2c)
```

Remember if you're using a board that doesn't support hardware I2C (like the ESP8266) you need to use the `bitbangio` module instead:

```
import board
import bitbangio
import adafruit_bno055
i2c = bitbangio.I2C(board.SCL, board.SDA)
sensor = adafruit_bno055.BNO055(i2c)
```

Now you're ready to read values from the sensor using any of these properties:

- **temperature** - The sensor temperature in degrees Celsius.
- **accelerometer** - This is a 3-tuple of X, Y, Z axis accelerometer values in meters per second squared.
- **magnetometer** - This is a 3-tuple of X, Y, Z axis magnetometer values in microteslas.
- **gyroscope** - This is a 3-tuple of X, Y, Z axis gyroscope values in degrees per second.
- **euler** - This is a 3-tuple of orientation Euler angle values.
- **quaternion** - This is a 4-tuple of orientation quaternion values.
- **linear_acceleration** - This is a 3-tuple of X, Y, Z linear acceleration values (i.e. without effect of gravity) in meters per second squared.
- **gravity** - This is a 3-tuple of X, Y, Z gravity acceleration values (i.e. without the effect of linear acceleration) in meters per second squared.

```
print('Temperature: {} degrees C'.format(sensor.temperature))
print('Accelerometer (m/s^2): {}'.format(sensor.accelerometer))
print('Magnetometer (microteslas): {}'.format(sensor.magnetometer))
print('Gyroscope (deg/sec): {}'.format(sensor.gyroscope))
print('Euler angle: {}'.format(sensor.euler))
print('Quaternion: {}'.format(sensor.quaternion))
print('Linear acceleration (m/s^2): {}'.format(sensor.linear_acceleration))
print('Gravity (m/s^2): {}'.format(sensor.gravity))
```

```

>>> print('Temperature: {}'.format(sensor.temperature))
Temperature: 21 degrees C
>>> print('Accelerometer (m/s^2): {}'.format(sensor.accelerometer))
Accelerometer (m/s^2): (-0.18, -0.13, 9.72)
>>> print('Magnetometer (microteslas): {}'.format(sensor.magnetometer))
Magnetometer (microteslas): (0.0, 0.0, 0.0)
>>> print('Gyroscope (deg/sec): {}'.format(sensor.gyroscope))
Gyroscope (deg/sec): (0.00222222, 0.00111111, 0.0)
>>> print('Euler angle: {}'.format(sensor.euler))
Euler angle: (359.938, -0.9375, -0.625)
>>> print('Quaternion: {}'.format(sensor.quaternion))
Quaternion: (0.999939, -0.00561523, 0.00823975, 0.0)
>>> print('Linear acceleration (m/s^2): {}'.format(sensor.linear_acceleration))
Linear acceleration (m/s^2): (0.02, 0.05, -0.07)
>>> print('Gravity (m/s^2): {}'.format(sensor.gravity))
Gravity (m/s^2): (-0.16, -0.1, 9.8)
>>>

```

That's all there is to using the BNO055 sensor with CircuitPython!

Here's a complete example that prints the acceleration and orientation (as Euler angles) every second. Save this as **main.py** on your board and look for the output in the serial REPL. Remember if using the ESP8266 and **bitbangio** module you need to change the initialization as mentioned above!

```

import board
import busio
import time

import adafruit_bno055

# Initialize I2C and the sensor.
i2c = busio.I2C(board.SCL, board.SDA)
sensor = adafruit_bno055.BNO055(i2c)

# Main loop runs forever printing acceleration and Euler angles every second.
while True:
    print('Accelerometer (m/s^2): {}'.format(sensor.accelerometer))
    print('Euler angle: {}'.format(sensor.euler))
    time.sleep(1.0)

```

Downloads

Files

- [Arduino Library](#)
- [EagleCAD PCB files on GitHub](#)
- [BNO055 Datasheet](#)
- [Fritzing object in the Adafruit Fritzing Library](#)

Pre-Compiled Bunny Rotate Binaries

The following binary images can be used in place of running the Processing Sketch, and may help avoid the frequent API and plugin changes.

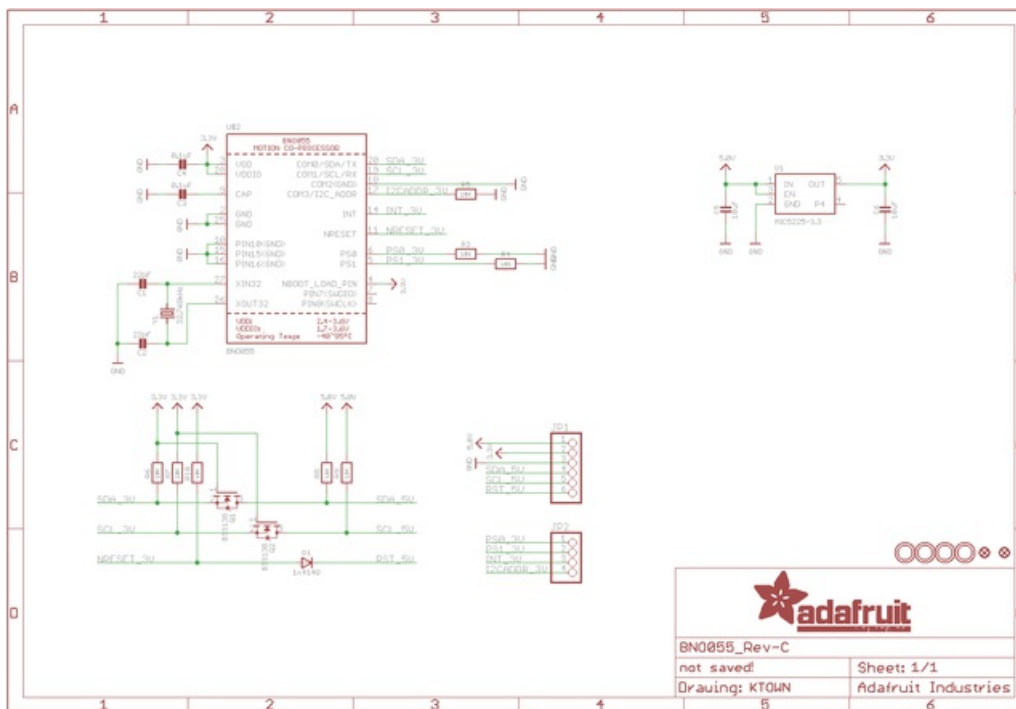
For OS X download **cuberotate.app.zip**, which was built on OS X 10.11.6 based on Processing 2.2.1:

cuberotate.app.zip

<https://adafru.it/wB4>

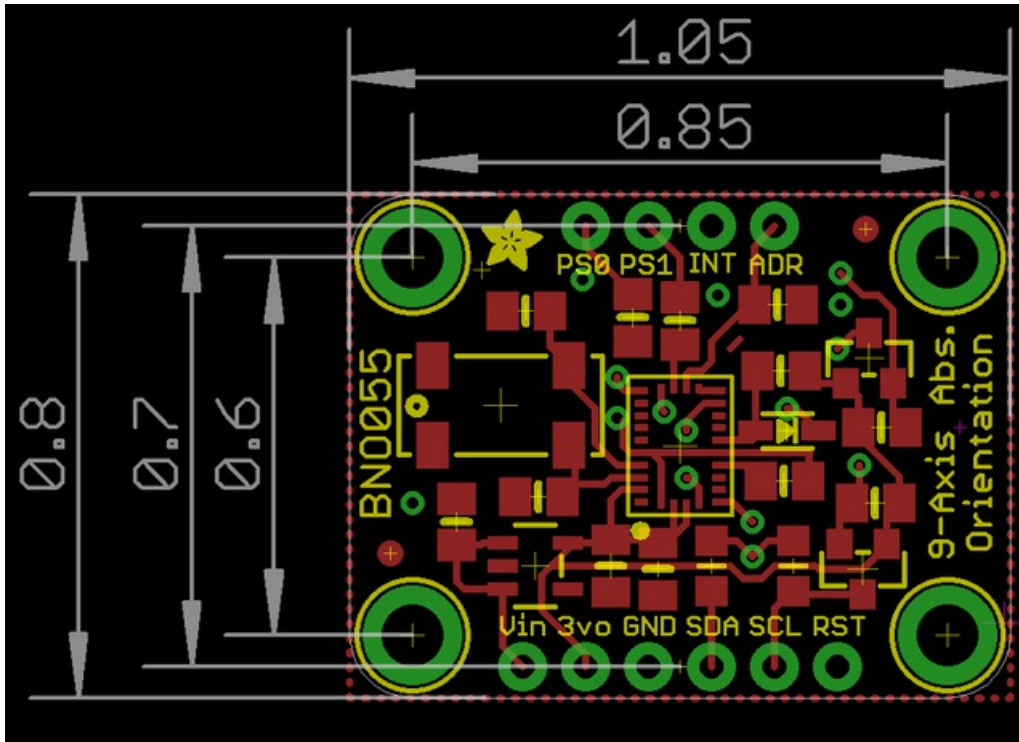
Schematic

The latest version of the Adafruit BNO055 breakout can be seen below (click the image to view the schematic in full resolution):



Board Dimensions

The BNO055 breakout has the following dimensions (in inches):



C.7 Ultrasonic Ranger Module HC - SR04



Tech Support: services@elecfreaks.com

Ultrasonic Ranging Module HC - SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:

- (1) Using IO trigger for at least 10us high level signal,
- (2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
- (3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.

Test distance = (high level time \times velocity of sound (340M/S) / 2,

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

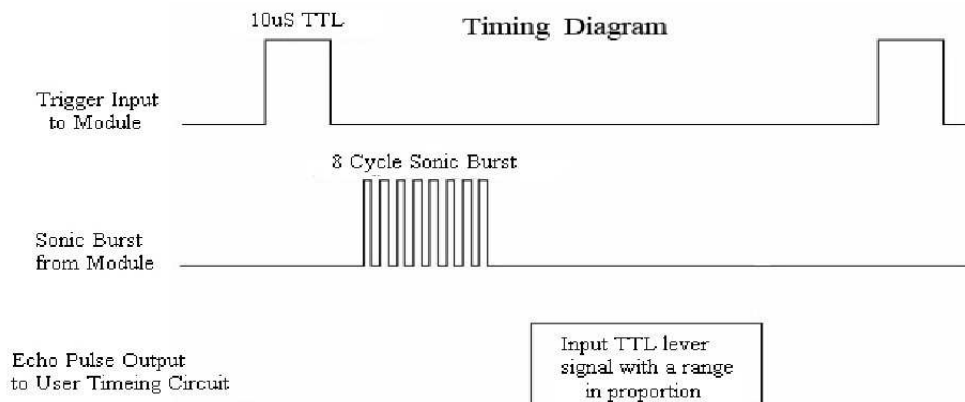
Electric Parameter

Working Voltage	DC 5 V
Working Current	15mA
Working Frequency	40Hz
Max Range	4m
Min Range	2cm
MeasuringAngle	15 degree
Trigger Input Signal	10uS TTL pulse
Echo Output Signal	Input TTL lever signal and the range in proportion
Dimension	45*20*15mm



Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion. You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: $\mu\text{S} / 58 = \text{centimeters}$ or $\mu\text{S} / 148 = \text{inch}$; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



Attention:

- The module is not suggested to connect directly to electric, if connected electric, the GND terminal should be connected the module first, otherwise, it will affect the normal work of the module.
- When tested objects, the range of area is not less than 0.5 square meters and the plane requests as smooth as possible, otherwise ,it will affect the results of measuring.

www.ElecFreaks.com



C.8 Lidar Lite v3



Lidar Lite v3 Operation Manual and Technical Specifications

Laser Safety

WARNING

This device requires no regular maintenance. In the event that the device becomes damaged or is inoperable, repair or service must be handled by authorized, factory-trained technicians only. Attempting to repair or service the unit on your own can result in direct exposure to laser radiation and the risk of permanent eye damage. For repair or service, contact your dealer or Garmin® for more information. This device should not be modified or operated without its housing or optics. Operating this device without a housing and optics, or operating this device with modified housing or optics that expose the laser source, may result in direct exposure to laser radiation and the risk of permanent eye damage. Removal or modification of the diffuser in front of the laser optic may result in the risk of permanent eye damage.

Use of controls or adjustments or performance of procedures other than those specified in this documentation may result in hazardous radiation exposure. Garmin is not responsible for injuries caused through the improper use or operation of this product.

CAUTION

This device emits laser radiation. This Laser Product is designated Class 1 during all procedures of operation. This designation means that the laser is safe to look at with the unaided eye, however it is advisable to avoid looking into the beam when operating the device and to turn off the module when not in use.

Documentation Revision Information

Rev	Date	Changes
0A	09/2016	Initial release

Table of Contents

Lidar Lite v3 Operation Manual and Technical Specifications	1
Laser Safety	1
Documentation Revision Information.....	1
Specifications	2
Physical	2
Electrical	2
Performance	2
Interface	2
Laser.....	2
Connections.....	2
Wiring Harness	2
Connector	2
Connector Port Identification	2
I2C Connection Diagrams	3
Standard I2C Wiring	3
Standard Arduino I2C Wiring	3
PWM Wiring.....	3
PWM Arduino Wiring.....	3
Operational Information.....	4
Technology	4
Theory of Operation.....	4
Interface.....	4
Initialization	4
Power Enable Pin	4
I2C Interface	4
Mode Control Pin	4
Settings.....	4
I2C Protocol Information.....	6
I2C Protocol Operation	7
Register Definitions	7
Control Register List	7
Detailed Control Register Definitions.....	8
Frequently Asked Questions.....	12
Must the device run on 5 Vdc? Can it run on 3.3 Vdc instead?.....	12
What is the spread of the laser beam?.....	12
How do distance, target size, aspect, and reflectivity effect returned signal strength?.....	12
How does the device work with reflective surfaces?	12
Diffuse Reflective Surfaces.....	12
Specular Surfaces	12
How does liquid affect the signal?	13

Specifications

Physical

Specification	Measurement
Size (LxWxH)	20 × 48 × 40 mm (0.8 × 1.9 × 1.6 in.)
Weight	22 g (0.78 oz.)
Operating temperature	-20 to 60°C (-4 to 140°F)

Electrical

Specification	Measurement
Power	5 Vdc nominal 4.5 Vdc min., 5.5 Vdc max.
Current consumption	105 mA idle 135 mA continuous operation

Performance

Specification	Measurement
Range (70% reflective target)	40 m (131 ft)
Resolution	+/- 1 cm (0.4 in.)
Accuracy < 5 m	±2.5 cm (1 in.) typical*
Accuracy ≥ 5 m	±10 cm (3.9 in.) typical Mean ±1% of distance maximum Ripple ±1% of distance maximum
Update rate (70% Reflective Target)	270 Hz typical 650 Hz fast mode** >1000 Hz short range only
Repetition rate	~50 Hz default 500 Hz max

*Nonlinearity present below 1 m (39.4 in.)

**Reduced sensitivity

Interface

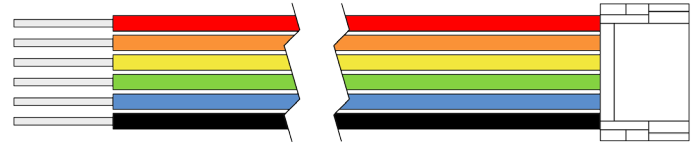
Specification	Measurement
User interface	I2C PWM External trigger
I2C interface	Fast-mode (400 kbit/s) Default 7-bit address 0x62 Internal register access & control
PWM interface	External trigger input PWM output proportional to distance at 10 µs/cm

Laser

Specification	Measurement
Wavelength	905 nm (nominal)
Total laser power (peak)	1.3 W
Mode of operation	Pulsed (256 pulse max. pulse train)
Pulse width	0.5 µs (50% duty Cycle)
Pulse train repetition frequency	10-20 KHz nominal
Energy per pulse	<280 nJ
Beam diameter at laser aperture	12 × 2 mm (0.47 × 0.08 in.)
Divergence	8 mRadian

Connections

Wiring Harness



Wire Color	Function
Red	5 Vdc (+)
Orange	Power enable (internal pull-up)
Yellow	Mode control
Green	I2C SCL
Blue	I2C SDA
Black	Ground (-)

There are two basic configurations for this device:

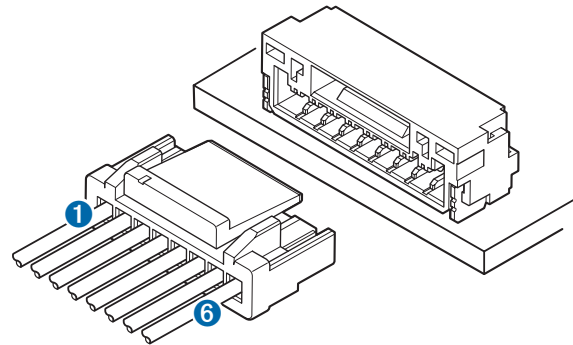
- **I2C (Inter-Integrated Circuit)**—a serial computer bus used to communicate between this device and a microcontroller, such as an Arduino board ([“I2C Interface”, page 4](#)).
- **PWM (Pulse Width Modulation)**—a bi-directional signal transfer method that triggers acquisitions and returns distance measurements using the mode-control pin ([“Mode Control Pin”, page 4](#)).

Connector

You can create your own wiring harness if needed for your project or application. The needed components are readily available from many suppliers.

Part	Description	Manufacturer	Part Number
Connector housing	6-position, rectangular housing, latch-lock connector receptacle with a 1.25 mm (0.049 in.) pitch.	JST	GHR-06V-S
Connector terminal	26-30 AWG crimp socket connector terminal (up to 6)	JST	SSHL-002T-P0.2
Wire	UL 1061 26 AWG stranded copper	N/A	N/A

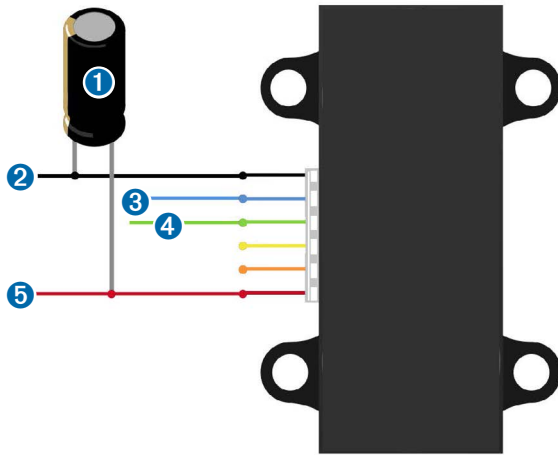
Connector Port Identification



Item	Pin	Function
1	1	5 Vdc (+)
	2	Power enable (internal pull-up)
	3	Mode control
	4	I2C SCL
	5	I2C SDA
6	6	Ground (-)

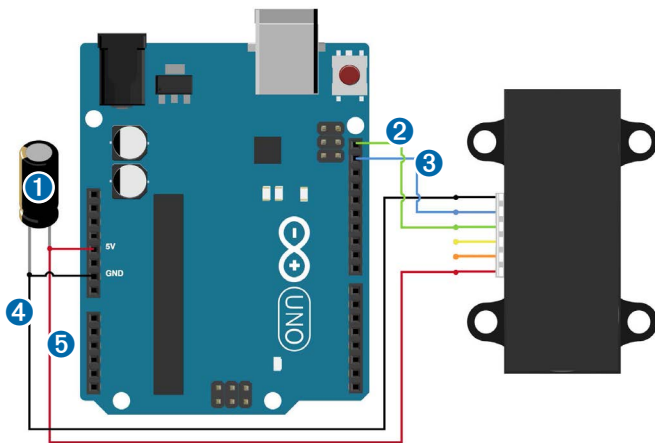
I2C Connection Diagrams

Standard I2C Wiring



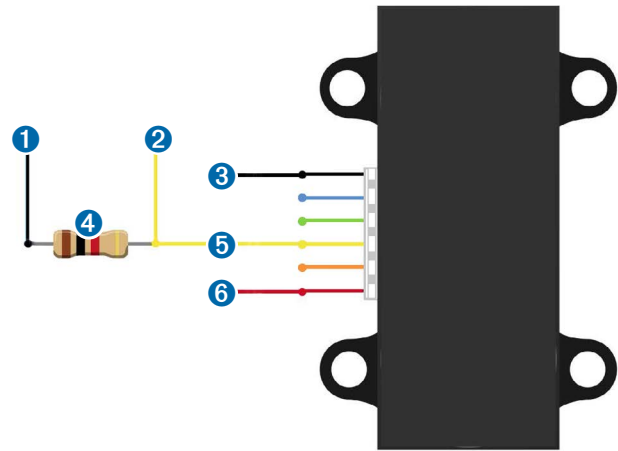
Item	Description	Notes
1	680µF electrolytic capacitor	You must observe the correct polarity when installing the capacitor.
2	Power ground (-) connection	Black wire
3	I2C SDA connection	Blue wire
4	I2C SCA connection	Green wire
5	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

Standard Arduino I2C Wiring



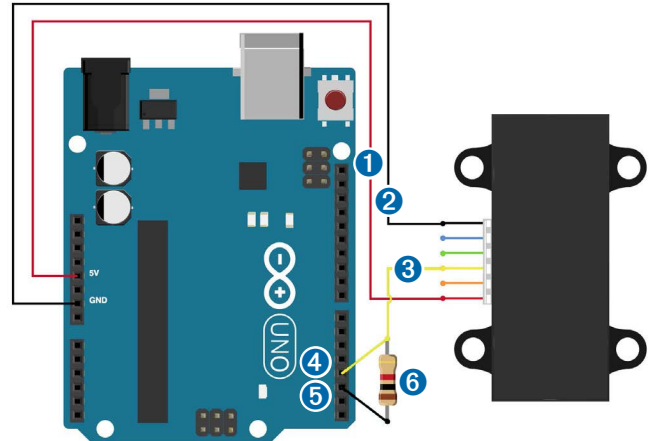
Item	Description	Notes
1	680µF electrolytic capacitor	You must observe the correct polarity when installing the capacitor.
2	I2C SCA connection	Green wire
3	I2C SDA connection	Blue wire
4	Power ground (-) connection	Black wire
5	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

PWM Wiring



Item	Description	Notes
1	Trigger pin on microcontroller	Connect the other side of the resistor to the trigger pin on your microcontroller.
2	Monitor pin on microcontroller	Connect one side of the resistor to the mode-control connection on the device, and to a monitoring pin on your microcontroller.
3	Power ground (-) connection	Black Wire
4	1kΩ resistor	
5	Mode-control connection	Yellow wire
6	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

PWM Arduino Wiring



Item	Description	Notes
1	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.
2	Power ground (-) connection	Black Wire
3	Mode-control connection	Yellow wire
4	Monitor pin on microcontroller	Connect one side of the resistor to the mode-control connection on the device, and to a monitoring pin on your microcontroller.
5	Trigger pin on microcontroller	Connect the other side of the resistor to the trigger pin on your microcontroller.
6	1kΩ resistor	

Operational Information

Technology

This device measures distance by calculating the time delay between the transmission of a Near-Infrared laser signal and its reception after reflecting off of a target. This translates into distance using the known speed of light. Our unique signal processing approach transmits a coded signature and looks for that signature in the return, which allows for highly effective detection with eye-safe laser power levels. Proprietary signal processing techniques are used to achieve high sensitivity, speed, and accuracy in a small, low-power, and low-cost system

Theory of Operation

To take a measurement, this device first performs a receiver bias correction routine, correcting for changing ambient light levels and allowing maximum sensitivity.

Then the device sends a reference signal directly from the transmitter to the receiver. It stores the transmit signature, sets the time delay for "zero" distance, and recalculates this delay periodically after several measurements.

Next, the device initiates a measurement by performing a series of acquisitions. Each acquisition is a transmission of the main laser signal while recording the return signal at the receiver. If there is a signal match, the result is stored in memory as a correlation record. The next acquisition is summed with the previous result. When an object at a certain distance reflects the laser signal back to the device, these repeated acquisitions cause a peak to emerge, out of the noise, at the corresponding distance location in the correlation record.

The device integrates acquisitions until the signal peak in the correlation record reaches a maximum value. If the returned signal is not strong enough for this to occur, the device stops at a predetermined maximum acquisition count.

Signal strength is calculated from the magnitude of the signal record peak and a valid signal threshold is calculated from the noise floor. If the peak is above this threshold the measurement is considered valid and the device will calculate the distance, otherwise it will report 1 cm. When beginning the next measurement, the device clears the signal record and starts the sequence again.

Interface

Initialization

On power-up or reset, the device performs a self-test sequence and initializes all registers with default values. After roughly 22 ms distance measurements can be taken with the I2C interface or the Mode Control Pin.

Power Enable Pin

The enable pin uses an internal pullup resistor, and can be driven low to shut off power to the device.

I2C Interface

This device has a 2-wire, I2C-compatible serial interface (refer to I2C-Bus Specification, Version 2.1, January 2000, available from Philips Semiconductor). It can be connected to an I2C bus as a slave device, under the control of an I2C master device. It supports 400 kHz Fast Mode data transfer.

The I2C bus operates internally at 3.3 Vdc. An internal level shifter allows the bus to run at a maximum of 5 Vdc. Internal 3k ohm pullup resistors ensure this functionality and allow for a simple connection to the I2C host.

The device has a 7-bit slave address with a default value of 0x62. The effective 8-bit I2C address is 0xC4 write and 0xC5 read. The device will not respond to a general call. Support is not provided for 10-bit addressing.

Setting the most significant bit of the I2C address byte to one triggers automatic incrementing of the register address with successive reads or writes within an I2C block transfer. This is commonly used to read the two bytes of a 16-bit value within one transfer and is used in the following example.

The simplest method of obtaining measurement results from the I2C interface is as follows:

- 1 Write 0x04 to register 0x00.
- 2 Read register 0x01. Repeat until bit 0 (LSB) goes low.
- 3 Read two bytes from 0x8f (High byte 0x0f then low byte 0x10) to obtain the 16-bit measured distance in centimeters.

A list of all available control registers is available on [page 7](#).

For more information about the I2C protocol, see [I2C Protocol Operation \(page 7\)](#).

Mode Control Pin

The mode control pin provides a means to trigger acquisitions and return the measured distance via Pulse Width Modulation (PWM) without having to use the I2C interface.

The idle state of the mode control pin is high impedance (High-Z). Pulling the mode control pin low will trigger a single measurement, and the device will respond by driving the line high with a pulse width proportional to the measured distance at 10 μ s/cm. A 1k ohm termination resistance is required to prevent bus contention.

The device drives the mode control pin high at 3.3 Vdc. Diode isolation allows the pin to tolerate a maximum of 5 Vdc.

As shown in the diagram [PWM Arduino Wiring \(page 3\)](#), a simple triggering method uses a 1k ohm resistor in series with a host output pin to pull the mode control pin low to initiate a measurement, and a host input pin connected directly to monitor the low-to-high output pulse width.

If the mode control pin is held low, the acquisition process will repeat indefinitely, producing a variable frequency output proportional to distance.

The mode control pin behavior can be modified with the ACQ_CONFIG_REG (0x04) I2C register as detailed in [0x04 \(page 8\)](#).

Settings

The device can be configured with alternate parameters for the distance measurement algorithm. This can be used to customize performance by enabling configurations that allow choosing between speed, range and sensitivity. Other useful features are also detailed in this section. See the full register map ([Control Register List \(page 7\)](#)) for additional settings not mentioned here.

Receiver Bias Correction

Address	Name	Description	Initial Value
0x00	ACQ_COMMAND	Device command	--

- Write 0x00: Reset device, all registers return to default values
- Write 0x03: Take distance measurement without receiver bias correction
- Write 0x04: Take distance measurement with receiver bias correction

Faster distance measurements can be performed by omitting the receiver bias correction routine. Measurement accuracy and sensitivity are adversely affected if conditions change (e.g. target distance, device temperature, and optical noise). To achieve good performance at high measurement rates receiver bias correction must be performed periodically. The recommended method is to do so at the beginning of every 100 sequential measurement commands.

Maximum Acquisition Count

Address	Name	Description	Initial Value
0x02	SIG_COUNT_VAL	Maximum acquisition count	0x80

The maximum acquisition count limits the number of times the device will integrate acquisitions to find a correlation record peak (from a returned signal), which occurs at long range or with low target reflectivity. This controls the minimum measurement rate and maximum range. The unit-less relationship is roughly as follows: rate = 1/n and range = $n^{(1/4)}$, where n is the number of acquisitions.

Measurement Quick Termination Detection

Address	Name	Description	Initial Value
0x04	ACQ_CONFIG_REG	Acquisition mode control	0x08

You can enable quick-termination detection by clearing bit 3 in this register. The device will terminate a distance measurement early if it anticipates that the signal peak in the correlation record will reach maximum value. This allows for faster and slightly less accurate operation at strong signal strengths without sacrificing long range performance.

Detection Sensitivity

Address	Name	Description	Initial Value
0x1c	THRESHOLD_BYPASS	Peak detection threshold bypass	0x00

The default valid measurement detection algorithm is based on the peak value, signal strength, and noise in the correlation record. This can be overridden to become a simple threshold criterion by setting a non-zero value. Recommended non-default values are 0x20 for higher sensitivity with more frequent erroneous measurements, and 0x60 for reduced sensitivity and fewer erroneous measurements.

Burst Measurements and Free Running Mode

Address	Name	Description	Initial Value
0x04	ACQ_CONFIG_REG	Acquisition mode control	0x08
0x11	OUTER_LOOP_COUNT	Burst measurement count control	0x00
0x45	MEASURE_DELAY	Delay between automatic measurements	0x14

The device can be configured to take multiple measurements for each measurement command or repeat indefinitely for free running mode.

OUTER_LOOP_COUNT (0x11) controls the number of times the device will retrigger itself. Values 0x00 or 0x01 result in the default one measurement per command. Values 0x02 to 0xfe directly set the repetition count. Value 0xff will enable free running mode after the host device sends an initial measurement command.

The default delay between automatic measurements corresponds to a 10 Hz repetition rate. Set bit 5 in ACQ_CONFIG_REG (0x04) to use the delay value in MEASURE_DELAY (0x45) instead. A delay value of 0x14 roughly corresponds to 100Hz.

The delay is timed from the completion of each measurement. The means that measurement duration, which varies with returned signal strength, will affect the repetition rate. At low repetition rates (high delay) this effect is small, but for lower delay values it is recommended to limit the maximum acquisition count if consistent frequency is desired.

Velocity

Address	Name	Description	Initial Value
0x09	VELOCITY	Velocity measurement output	--

The velocity measurement is the difference between the current measurement and the previous one, resulting in a signed (2's complement) 8-bit number in cm. Positive velocity is away from the device. This can be combined with free running mode for a constant measurement frequency. The default free running frequency of 10 Hz therefore results in a velocity measurement in .1 m/s.

Configurable I2C Address

Address	Name	Description	Initial Value
0x16	UNIT_ID_HIGH	Serial number high byte	Unique
0x17	UNIT_ID_LOW	Serial number low byte	Unique
0x18	I2C_ID_HIGH	Write serial number high byte for I2C address unlock	--
0x19	I2C_ID_LOW	Write serial number low byte for I2C address unlock	--
0x1a	I2C_SEC_ADDR	Write new I2C address after unlock	--
0x1e	I2C_CONFIG	Default address response control	0x00

The I2C address can be changed from its default value. Available addresses are 7-bit values with a '0' in the least significant bit (even hexadecimal numbers).

To change the I2C address, the unique serial number of the unit must be read then written back to the device before setting the new address. The process is as follows:

- 1 Read the two byte serial number from 0x96 (High byte 0x16 and low byte 0x17).
- 2 Write the serial number high byte to 0x18.
- 3 Write the serial number low byte to 0x19.
- 4 Write the desired new I2C address to 0x1a.
- 5 Write 0x08 to 0x1e to disable the default address.

This can be used to run multiple devices on a single bus, by enabling one, changing its address, then enabling the next device and repeating the process.

The I2C address will be restored to default after a power cycle.

Power Control

Address	Name	Description	Initial Value
0x65	POWER_CONTROL	Power state control	0x80

NOTE: The most effective way to control power usage is to utilize the enable pin to deactivate the device when not in use.

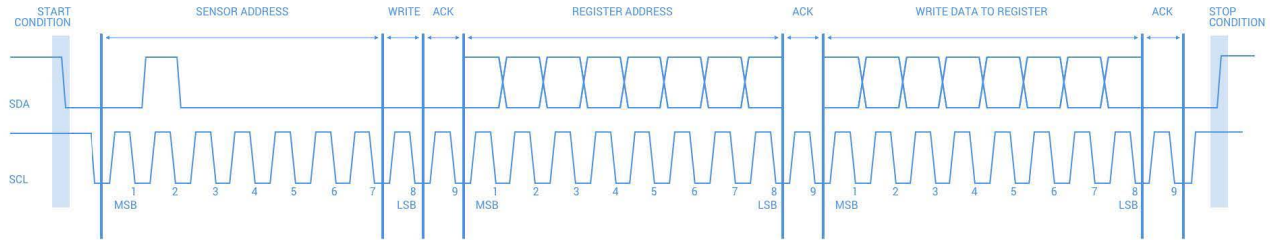
Another option is to set bit 0 in this register which disables the receiver circuit, saving roughly 40mA. After being re-enabled, the receiver circuit stabilizes by the time a measurement can be performed. Setting bit 2 puts the device in sleep mode until the next I2C transaction, saving 20mA. **Since the wake-up time is only around 2 m/s shorter than the full power-on time, and both will reset all registers, it is recommended to use the enable pin instead.**

I2C Protocol Information

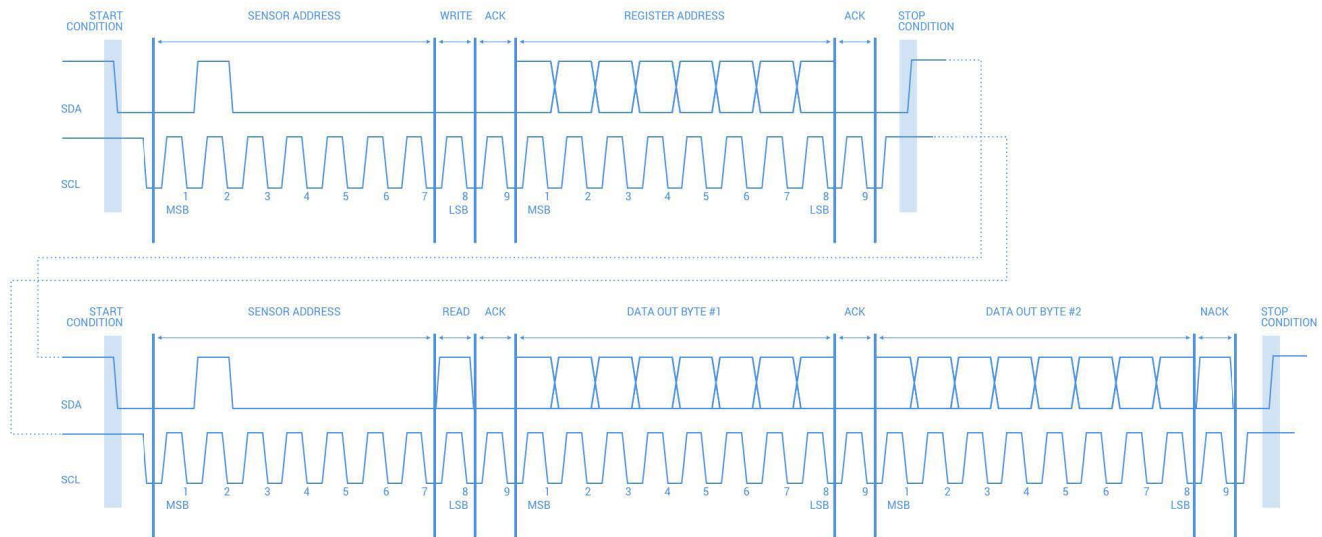
This device has a 2-wire, I2C-compatible serial interface (refer to I2C-Bus Specification, Version 2.1, January 2000, available from Philips Semiconductor). It can be connected to an I2C bus as a slave device, under the control of an I2C master device. It supports standard 400 kHz data transfer mode. Support is not provided for 10-bit addressing.

The Sensor module has a 7-bit slave address with a default value of 0x62 in hexadecimal notation. The effective 8 bit I2C address is: 0xC4 write, 0xC5 read. The device will not presently respond to a general call.

Write



Read



Notes:

- This device does not work with repeated START conditions. It must first receive a STOP condition before a new START condition.
- The ACK and NACK items are responses from the master device to the slave device.
- The last NACK in the read is technically optional, but the formal I2C protocol states that the master shall not acknowledge the last byte.

I2C Protocol Operation

The I2C serial bus protocol operates as follows:

- 1 The master initiates data transfer by establishing a start condition, which is when a high-to-low transition on the SDA line occurs while SCL is high. The following byte is the address byte, which consists of the 7-bit slave address followed by a read/write bit with a zero state indicating a write request. A write operation is used as the initial stage of both read and write transfers. If the slave address corresponds to the module's address the unit responds by pulling SDA low during the ninth clock pulse (this is termed the acknowledge bit). At this stage, all other devices on the bus remain idle while the selected device waits for data to be written to or read from its shift register.
- 2 Data is transmitted over the serial bus in sequences of nine clock pulses (eight data bits followed by an acknowledge bit). The transitions on the SDA line must occur during the low period of SCL and remain stable during the high period of SCL.
- 3 An 8 bit data byte following the address loads the I2C control register with the address of the first control register to be read along with flags indicating if auto increment of the addressed control register is desired with successive reads or writes; and if access to the internal micro or external correlation processor register space is requested. Bit locations 5:0 contain the control register address while bit 7 enables the automatic incrementing of control register with successive data blocks. Bit position 6 selects correlation memory external to the microcontroller if set. (Presently an advanced feature)
- 4 If a read operation is requested, a stop bit is issued by the master at the completion of the first data frame followed by the initiation of a new start condition, slave address with the read bit set (one state). The new address byte is followed by the reading of one or more data bytes succession. After the slave has acknowledged receipt of a valid address, data read operations proceed by the master releasing the I2C data line SDA with continuing clocking of SCL. At the completion of the receipt of a data byte, the master must strobe the acknowledge bit before continuing the read cycle.
- 5 For a write operation to proceed, Step 3 is followed by one or more 8 bit data blocks with acknowledges provided by the slave at the completion of each successful transfer. At the completion of the transfer cycle a stop condition is issued by the master terminating operation.

Register Definitions

Control Register List

Address	R/W	Name	Description	Initial Value	Details
0x00	W	ACQ_COMMAND	Device command	--	page 8
0x01	R	STATUS	System status	--	page 8
0x02	R/W	SIG_COUNT_VAL	Maximum acquisition count	0x80	page 8
0x04	R/W	ACQ_CONFIG_REG	Acquisition mode control	0x08	page 8
0x09	R	VELOCITY	Velocity measurement output	--	page 8
0x0c	R	PEAK_CORR	Peak value in correlation record	--	page 8
0x0d	R	NOISE_PEAK	Correlation record noise floor	--	page 8
0x0e	R	SIGNAL_STRENGTH	Received signal strength	--	page 9
0x0f	R	FULL_DELAY_HIGH	Distance measurement high byte	--	page 9
0x10	R	FULL_DELAY_LOW	Distance measurement low byte	--	page 9
0x11	R/W	OUTER_LOOP_COUNT	Burst measurement count control	0x01	page 9
0x12	R/W	REF_COUNT_VAL	Reference acquisition count	0x05	page 9
0x14	R	LAST_DELAY_HIGH	Previous distance measurement high byte	--	page 9
0x15	R	LAST_DELAY_LOW	Previous distance measurement low byte	--	page 9
0x16	R	UNIT_ID_HIGH	Serial number high byte	Unique	page 9
0x17	R	UNIT_ID_LOW	Serial number low byte	Unique	page 9
0x18	W	I2C_ID_HIGH	Write serial number high byte for I2C address unlock	--	page 9
0x19	W	I2C_ID_LOW	Write serial number low byte for I2C address unlock	--	page 9
0x1a	R/W	I2C_SEC_ADDR	Write new I2C address after unlock	--	page 9
0x1c	R/W	THRESHOLD_BYPASS	Peak detection threshold bypass	0x00	page 9
0x1e	R/W	I2C_CONFIG	Default address response control	0x00	page 9
0x40	R/W	COMMAND	State command	--	page 10
0x45	R/W	MEASURE_DELAY	Delay between automatic measurements	0x14	page 10
0x4c	R	PEAK_BCK	Second largest peak value in correlation record	--	page 10
0x52	R	CORR_DATA	Correlation record data low byte	--	page 10
0x53	R	CORR_DATA_SIGN	Correlation record data high byte	--	page 10
0x5d	R/W	ACQ_SETTINGS	Correlation record memory bank select	--	page 10
0x65	R/W	POWER_CONTROL	Power state control	0x80	page 10

Detailed Control Register Definitions

NOTE: Unless otherwise noted, all registers contain one byte and are read and write.

0x00

R/W	Name	Description	Initial Value
W	ACQ_COMMAND	Device command	--

Bit	Function
7:0	Write 0x00: Reset FPGA, all registers return to default values Write 0x03: Take distance measurement without receiver bias correction Write 0x04: Take distance measurement with receiver bias correction

0x01

R/W	Name	Description	Initial Value
R	STATUS	System status	--

Bit	Function
6	Process Error Flag 0: No error detected 1: System error detected during measurement
5	Health Flag 0: Error detected 1: Reference and receiver bias are operational
4	Secondary Return Flag 0: No secondary return detected 1: Secondary return detected in correlation record
3	Invalid Signal Flag 0: Peak detected 1: Peak not detected in correlation record, measurement is invalid
2	Signal Overflow Flag 0: Signal data has not overflowed 1: Signal data in correlation record has reached the maximum value before overflow. This occurs with a strong received signal strength
1	Reference Overflow Flag 0: Reference data has not overflowed 1: Reference data in correlation record has reached the maximum value before overflow. This occurs periodically
0	Busy Flag 0: Device is ready for new command 1: Device is busy taking a measurement

0x02

R/W	Name	Description	Initial Value
R/W	SIG_COUNT_VAL	Maximum acquisition count	0x80

Bit	Function
7:0	Maximum number of acquisitions during measurement

0x04

R/W	Name	Description	Initial Value
R/W	ACQ_CONFIG_REG	Acquisition mode control	0x08

Bit	Function
6	0: Enable reference process during measurement 1: Disable reference process during measurement
5	0: Use default delay for burst and free running mode 1: Use delay from MEASURE_DELAY (0x45) for burst and free running mode
4	0: Enable reference filter, averages 8 reference measurements for increased consistency 1: Disable reference filter
3	0: Enable measurement quick termination. Device will terminate distance measurement early if it anticipates that the signal peak in the correlation record will reach maximum value. 1: Disable measurement quick termination.
2	0: Use default reference acquisition count of 5. 1: Use reference acquisition count from REF_COUNT_VAL (0x12).
1:0	Mode Select Pin Function Control 00: Default PWM mode. Pull pin low to trigger measurement, device will respond with an active high output with a duration of 10us/cm. 01: Status output mode. Device will drive pin active high while busy. Can be used to interrupt host device. 10: Fixed delay PWM mode. Pulling pin low will not trigger a measurement. 11: Oscillator output mode. Nominal 31.25 kHz output. The accuracy of the silicon oscillator in the device is generally within 1% of nominal. This affects distance measurements proportionally and can be measured to apply a compensation factor.

0x09

R/W	Name	Description	Initial Value
R	VELOCITY	Velocity measurement output	--

Bit	Function
7:0	Velocity measurement output. The difference between the current measurement and the previous one, signed (2's complement) value in centimeters.

0x0c

R/W	Name	Description	Initial Value
R	PEAK_CORR	Peak value in correlation record	--

Bit	Function
7:0	The value of the highest peak in the correlation record.

0x0d

R/W	Name	Description	Initial Value
R	NOISE_PEAK	Correlation record noise floor	--

Bit	Function
7:0	A measure of the noise in the correlation record. Will be slightly above the third highest peak.

0x0e

R/W	Name	Description	Initial Value
R	SIGNAL_STRENGTH	Received signal strength	--

Bit	Function
7:0	Received signal strength calculated from the value of the highest peak in the correlation record and how many acquisitions were performed.

0x0f

R/W	Name	Description	Initial Value
R	FULL_DELAY_HIGH	Distance measurement high byte	--

Bit	Function
7:0	Distance measurement result in centimeters, high byte.

0x10

R/W	Name	Description	Initial Value
R	FULL_DELAY_LOW	Distance measurement low byte	--

Bit	Function
7:0	Distance measurement result in centimeters, low byte.

0x11

R/W	Name	Description	Initial Value
R/W	OUTER_LOOP_COUNT	Burst measurement count control	0x01

Bit	Function
7:0	0x00-0x01: One measurement per distance measurement command. 0x02-0xfe: Repetition count per distance measurement command. 0xff: Indefinite repetitions after initial distance measurement command. See ACQ_CONFIG_REG (0x04) and MEASURE_DELAY (0x45) for non-default automatic repetition delays.

0x12

R/W	Name	Description	Initial Value
R/W	REF_COUNT_VAL	Reference acquisition count	0x05

Bit	Function
7:0	Non-default number of reference acquisitions during measurement. ACQ_CONFIG_REG (0x04) bit 2 must be set.

0x14

R/W	Name	Description	Initial Value
R	LAST_DELAY_HIGH	Previous distance measurement high byte	--

Bit	Function
7:0	Previous distance measurement result in centimeters, high byte.

0x15

R/W	Name	Description	Initial Value
R	LAST_DELAY_LOW	Previous distance measurement low byte	--

Bit	Function
7:0	Previous distance measurement result in centimeters, low byte.

0x16

R/W	Name	Description	Initial Value
R	UNIT_ID_HIGH	Serial number high byte	Unique

Bit	Function
7:0	Unique serial number of device, high byte.

0x17

R/W	Name	Description	Initial Value
R	UNIT_ID_LOW	Serial number low byte	Unique

Bit	Function
7:0	Unique serial number of device, high byte.

0x18

R/W	Name	Description	Initial Value
W	I2C_ID_HIGH	Write serial number high byte for I2C address unlock	--

Bit	Function
7:0	Write the value in UNIT_ID_HIGH (0x16) here as part of enabling a non-default I2C address. See I2C_ID_LOW (0x19) and I2C_SEC_ADDR (0x1a).

0x19

R/W	Name	Description	Initial Value
W	I2C_ID_LOW	Write serial number low byte for I2C address unlock	--

Bit	Function
7:0	Write the value in UNIT_ID_LOW (0x17) here as part of enabling a non-default I2C address. See I2C_ID_HIGH (0x18) and I2C_SEC_ADDR (0x1a).

0x1a

R/W	Name	Description	Initial Value
R/W	I2C_SEC_ADDR	Write new I2C address after unlock	--

Bit	Function
7:0	Non-default I2C address. Available addresses are 7-bit values with a '0' in the least significant bit (even hexadecimal numbers). I2C_ID_HIGH (0x18) and I2C_ID_LOW (0x19) must have the correct value for the device to respond to the non-default I2C address.

0x1c

R/W	Name	Description	Initial Value
R/W	THRESHOLD_BYPASS	Peak detection threshold bypass	0x00

Bit	Function
7:0	0x00: Use default valid measurement detection algorithm based on the peak value, signal strength, and noise in the correlation record. 0x01-0xff: Set simple threshold for valid measurement detection. Values 0x20-0x60 generally perform well.

0x1e

R/W	Name	Description	Initial Value
R/W	I2C_CONFIG	Default address response control	0x00

Bit	Function
-----	----------

3	0: Device will respond to I2C address 0x62. Device will also respond to non-default address if configured successfully. See I2C_ID_HIGH (0x18), I2C_ID_LOW (0x19), and I2C_SEC_ADDR (0x1a). 1: Device will only respond to non-default I2C address. It is recommended to configure the non-default address first, then use the non-default address to write to this register, ensuring success.
---	--

0x40

R/W	Name	Description	Initial Value
R/W	COMMAND	State command	--

Bit	Function
2:0	000: Test mode disable, resume normal operation 111: Test mode enable, allows download of correlation record Select correlation memory bank in ACQ_SETTINGS (0x5d) prior to enabling test mode. Once test mode is enabled, read CORR_DATA (0x52) and CORR_DATA_SIGN (0x53) in one transaction (read from 0xd2). The memory index is incremented automatically and successive reads produce sequential data.

0x45

R/W	Name	Description	Initial Value
R/W	MEASURE_DELAY	Delay between automatic measurements	0x14

Bit	Function
7:0	Non-default delay after completion of measurement before automatic retrigger, in burst and continuous modes. ACQ_CONFIG_REG (0x04) bit 5 must be set. Value 0xc8 corresponds to 10 Hz repetition rate and 0x14 to roughly 100 Hz.

0x4c

R/W	Name	Description	Initial Value
R	PEAK_BCK	Second largest peak value in correlation record	--

Bit	Function
7:0	The value of the second highest peak in the correlation record.

0x52

R/W	Name	Description	Initial Value
R	CORR_DATA	Correlation record data low byte	--

Bit	Function
7:0	Correlation record data low byte. See CORR_DATA_SIGN (0x53), ACQ_SETTINGS (0x5d), and COMMAND (0x40).

0x53

R/W	Name	Description	Initial Value
R	CORR_DATA_SIGN	Correlation record data high byte	--

Bit	Function
7:0	Correlation record data high byte. Correlation record data is a 2's complement 9-bit value, and must be sign extended to be formatted as a 16-bit 2's complement value. Thus when repacking the two bytes obtained for the I2C transaction, set the high byte to 0xff if the LSB of the high byte is one.

0x5d

R/W	Name	Description	Initial Value
R/W	ACQ_SETTINGS	Correlation record memory bank select	--

Bit	Function
7:6	11: Access correlation memory bank. Write prior to test mode enable, see COMMAND (0x40).

0x65

R/W	Name	Description	Initial Value
R/W	POWER_CONTROL	Power state control	0x80

Bit	Function
2	1: Device Sleep, wakes upon I2C transaction. Registers are reinitialized, wakeup time similar to full reset using enable pin. 0: Device awake
0	1: Disable receiver circuit 0: Enable receiver circuit. Receiver circuit stabilizes by the time a measurement can be performed.