Arduino Controllers for the Microwave Operator

Greg McIntire, AA5C

Arduino and Raspberry PI microcontrollers have become popular and found widespread use in amateur radio applications. Over the past few years it seems almost every issue of QST has an article on an Arduino application, with good reason. They are versatile and low-cost with a variety of low-cost, easy to interface hardware add-ons and a wealth of open-source software available. I became interested in Arduino controllers as a means to control phase locked loops (PLL) and direct digital synthesizers (DDS) that use a digital control interface and have experimented with a number of PLL and DDS using Arduino controllers. I reported on the performance of several low-cost PLL evaluation boards based on the Analog Devices ADF4351 and ADF5355 PLLs, as well as other projects, in a 2017 Microwave Update article (Reference 1). This 2019 article reports on the results of several Arduino-based projects since that time and includes general information about the controllers and available supporting hardware intended to help you apply them to you needs.

Most modern PLLs use a serial peripheral interface (SPI) bus for control and status. This makes control options robust but also requires some sort of controller in order to use the part since a number of registers need to be loaded in order to make the parts function. PLL and DDS register lengths are now generally 32-bits which make for easy code reuse. PLLs with other length registers can be controlled as well. I had an older (circa 1996) National LMX2316 PLL evaluation board that I wanted to get running. Three 21-bit registers need to be loaded for it to operate. Coding turned out to be simple since the first 11 bits of each 32-bit register transfer are serially shifted into the "bit bucket" and the last 21-bits containing the necessary control bits end up in the correct position. The hardware interface was also the simple four-wire SPI interface, and since the LM2316XL uses 5 volt logic, no level shifting resistors were required to interface it to the Arduino Uno. This particular evaluation board is older and has a limited frequency range compared to current devices, but the project illustrates how an Arduino can be used on a range of different PLLs (Figure 1 LMX2316 Evaluation Board Controlled by an Arduino Uno).

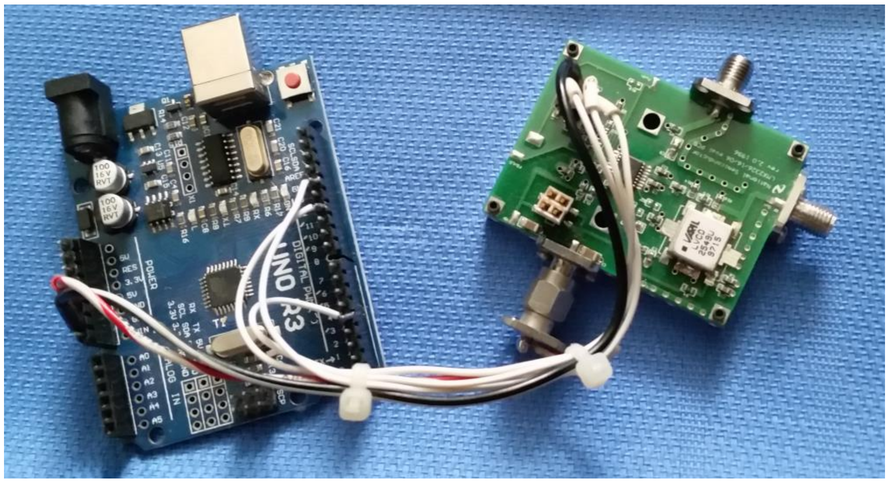

Figure 1 LMX2316 Evaluation Board Controlled by an Arduino Uno

Studying the register set of each particular PLL is the best way to understand the capabilities and limitations of each device. Current PLLs have many features you can exploit if you are willing to dig in to really understand the device. For instance, they can be programmed for best phase noise performance or best spur performance. Phase can be controlled on some to support the generation of complex waveforms. Some device manufacturers make software tools available that generate the register values for the particular frequencies and configurations you select. I'm familiar with several of the Analog Devices tools and they can greatly simply the generation of the control register values. However, I still go through the data sheets in detail to better understand the part. The software tools are generally useful for deriving control values for a single frequency. When you are changing frequencies on the fly in real-time, what I refer to as dynamic control, you need to understand the frequency control formulas sufficiently in order to program the controller. The same applies for other real-time controls like on-off keying of the RF output.

Some of the PLLs are programmable with frequency resolution down to a Hertz or so. These parts can tax or exceed the capabilities of some of the simpler controllers. The ADF5355 is a fractional-N PLL with the following frequency control register lengths that allow near Hz frequency resolution: INT 16-bits, FRAC1 24-bits, FRAC2 14-bits, and MOD2 14-bits. The length of these frequency control registers should give you an idea of the achievable resolution.

I've found that the simple 8-bit Arduino Uno works well for loading 32-bit register values in fixed-frequency LO applications where values are pre-calculated. My experience in calculating the values in real-time was that I could get the precision needed for the ADF4351 using an 8-bit Arduino Uno, but not for the ADF5355, even using extended length arithmetic. A 32-bit Arduino Due did the job nicely and my Arduino Due/ADF5355 combination has been a useful lab tool. I've had some brief email exchanges with Alain, F1CJN, and Ed, W7GLF and they may have worked out a means of using the Arduino Uno for dynamic control of the ADF5355 but I have not seen their code or technique to try it.

Recently I completed upgrading the six transverters in my microwave station (902 through 10368 MHz) with local oscillators (LO) locked to a GPS-disciplined 10 MHz reference. I started with Apollo 32 boards for 758 MHz, 1152 MHz, and 2160 MHz LOs. This was before

Figure 2 AA5C Microwave Transverters

I became aware of the ADF4351 evaluation boards. I then used an ADF4351 board for 3312 MHz and ADF5355 boards for 5616 MHz and 10224 MHz. All of the ADF4351 and ADF5355 boards are controlled by Arduino Unos. The BG7TBL GPS-disciplined 10 MHz oscillator (green faced box at the top of the picture) feeds a two-way power splitter which then feeds two 4-way splitter/amplifier/filter boards from Down East Microwave. A total of eight outputs are available. The boards are in the chassis shown next to the power meter at the top of the picture. Now each of the six microwave transverters in the shack and a microwave frequency counter are locked to GPS. Not that long ago we would sometimes have to tune up to +/-50 KHz on the IF rig to find the station we were trying to work. That operating variable is now significantly reduced or eliminated, depending on what the other station is using.

A single controller can be used to control multiple PLLs or devices that use a SPI bus interface. An additional I/O pin needs to be assigned for each extra device connected to the SPI bus for the load enable line (LE) and called out separately in the code. I recently completed a 47 GHz transverter that uses two ADF5355 evaluation boards for the LOs. It is uses a double conversion frequency plan. The two ADF5355 evaluation boards are controlled by one Arduino

Uno (Figure 3 AA5C 47 GHz Transverter) as shown in the lower left of the figure. The additional amplifiers needed to bring the relatively low (~ -8 dBm) output power levels of the ADF5355 evaluation boards (two 12 GHz amplifiers are required for the 12,978 MHz LO and one for the 8.01 GHz LO) can also be seen in the picture. The 32 MHz reference signal is supplied to each synthesizer from a two-way power splitter.

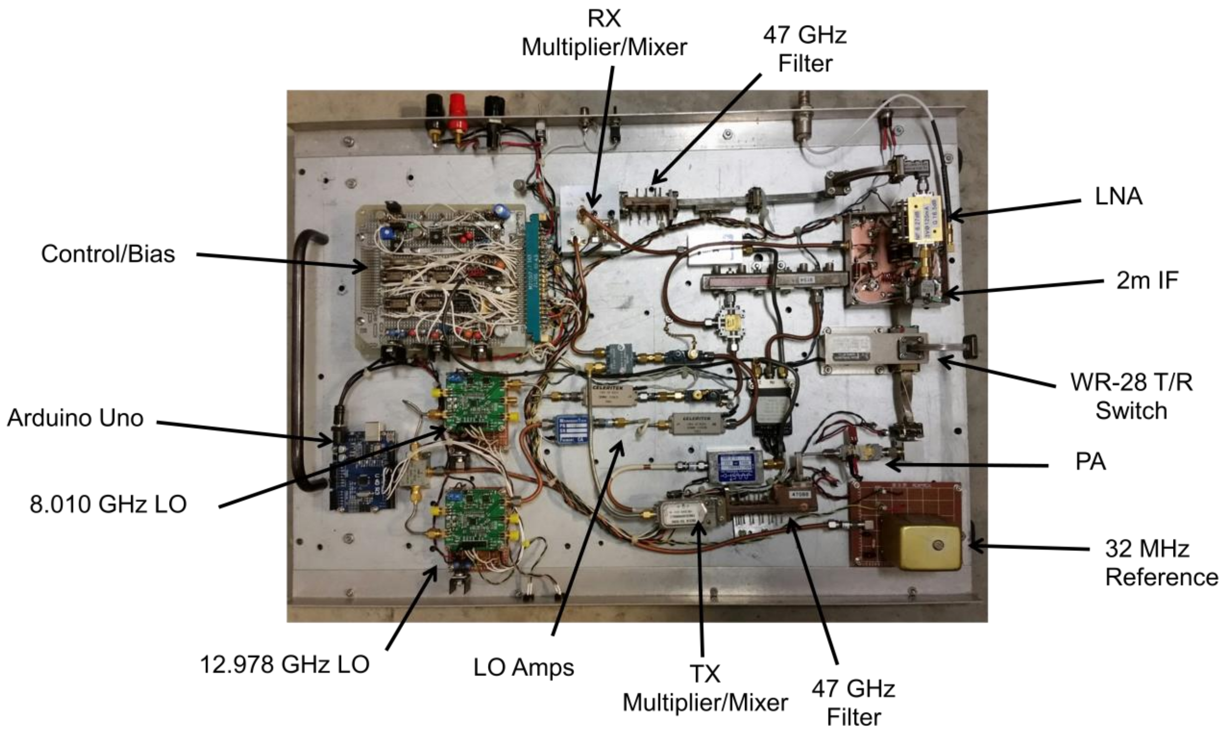

Figure 3 AA5C 47 GHz Transverter

The versatility of using a controller for the PLL is exemplified by the fact that I started the project with a 10 MHz reference. I couldn't calibrate the 10 MHz reference to precisely 10 MHz so I switched to a 32 MHz reference that I could calibrate precisely. A quick recalculation of the register values and flashing them into the controller was all that was required to use the new reference.

The frequency accuracy of the 47 GHz transverter has turned out to be quite good. It is not locked to a GPS-disciplined oscillator but I did calibrate the 32 MHz reference using a microwave counter locked to a GPS-disciplined oscillator. When working Al, W5LUA on 47 GHz, we have generally been within a KHz or two of each other which is pretty amazing at 47 GHz.

I've experimented with the four Arduino controllers shown in Figure 4 Example Arduino/WEMOS Controllers .   Clockwise, starting in the upper left hand corner of the figure and working clockwise, these are the Arduino Mega 2560, the Arduino Uno, the WEMOS D1 Mini, and the Arduino Due

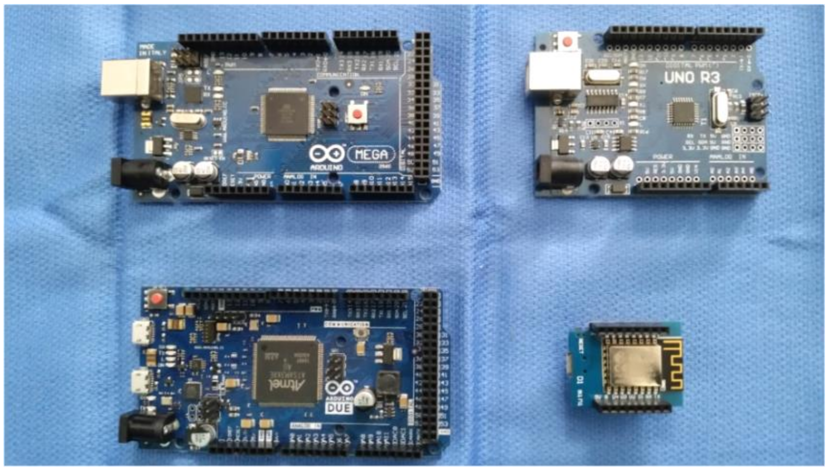

Figure 4 Example Arduino/WEMOS Controllers

Some of the key features of these controllers are tabulated in Table 1 Comparison of Controller Key Features. I extracted these values from the Arduino and WEMOS websites and believe they are accurate but encourage you to check for yourself. Hopefully this will give you a starting place for picking a particular controller for your application.   The more expensive ones were around $20. The Uno and D1 Mini can be found for $5 or less.

Board	**Mega 2560**	**Uno**	**Due**	**D1 Mini**
Processor	ATmega2560	ATmega328P	ATSAM3X8E	ESP8266
Clock Speed (MHz)	16	16*	84	80/160
Word Length (bits)	32	8	32	32
I/O Logic Voltage	3.3	5	3.3	3.3
Analog I/O (in/out)	16/0	6/0	12/2	1/0
Digital (I/O or PWM)	54/15	14/6	54/12	11
EEPROM (KB)	4	1	0	512 bytes
SRAM (KB)	8	2	96	4 MB
Flash (KB)	256	32	512	4 MB
SPI Bus Interfaces	1	1	2	1
USB Interface	1-Regular	1-Regular	2-Micro	1-Micro

Table 1 Comparison of Controller Key Features

*It is important to note that a lot of the microcontrollers sold are copies of the real Arduino.cc boards. The schematics are readily available online. They can be cheaper than Arduino boards but the details can vary so you need to be a bit careful depending on your application. For instance, the Uno R3 pictured above is a copy. I've found them to generally function like the Arduino Uno R3 but the clock/crystal on the copy is 12 MHz versus 16 MHz on the Arduino.cc board. You'll need to take this into account if the code you are using is clock speed dependent.

As noted above, I used an Arduino Due for dynamic control of an ADF5355 evaluation board. The first time I compiled the code for this project I found out the Arduino Due doesn't have any EEPROM. It wasn't hard to add external EEPROM and it is a good example of the low-cost hardware add-ons available. I found small EEPROM boards with an I2C interface for less than $2 each and adapted some open source code to interface it to the Due. Other examples of low-cost hardware you can get for your projects are shown in Figures 5 and 6. The LCD Display and Button Shield (Figure 5 2X16 LCD Display and Button Shield) plugs directly into the connectors on controllers like the Uno, Mega2560, and Due. This provides a handy human interface independent of a PC.

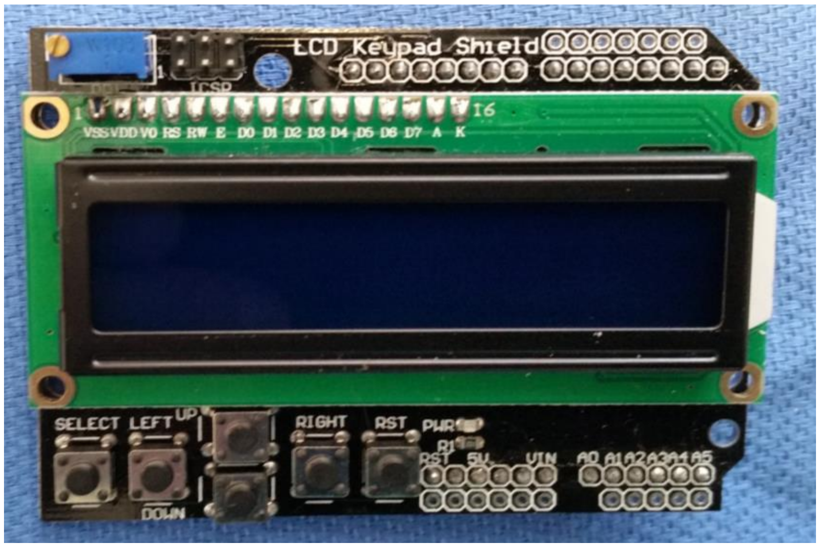

Figure 5 2X16 LCD Display and Button Shield

The relay boards shown in Figure 6 Arduino Compatible Relay Boards are handy for controlling devices not directly compatible with the controller I/O lines. The four-relay board on top is a shield and can plug directly onto the Mega2560, Uno, or Due, for instance. I did find a conflict on one of the I/O lines resulting in only three of the relays being useable when plugged into the controller for my application. The eight-relay board in the bottom of the figure is not a shield and although you don't have the convenience of a shield, you can assign whatever I/O pins you like to control the relays.

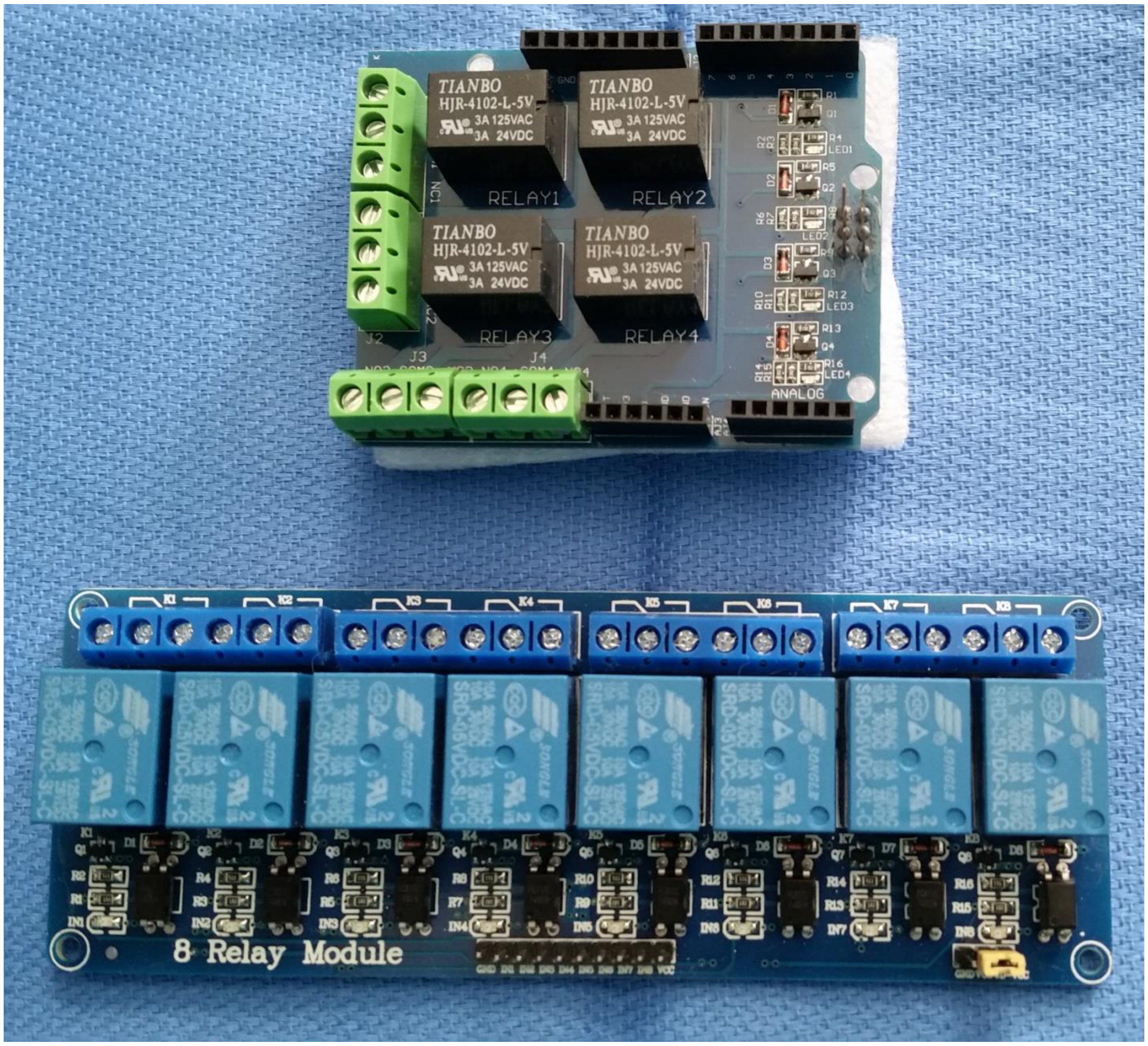

Figure 6 Arduino Compatible Relay Boards

There are many other devices that can be used with these controllers. I used the LM35 temperature sensor as part of the NTMS 432 MHz beacon project to send PA temperature as part of the beacon message. I've also experimented with LAN shields for local network use. An interesting note is that I found the software needed to go on the open internet (for virus protection, etc.) quickly gets too large for use with the smaller controllers. Prototyping shields, GPS modules, and other hardware too numerous to list are available for use with these controllers.

My final example is an in-work project using a 3-axis position sensor. The 3-axis sensor IC is a HMC5883L. I'm using the GY-271 version of the board and it connects to the controller via an I2C interface (Figure 7 3-axis Magnetic Sensor Interface to an Uno R3). The sensor is mounted to a small board and these can be found for just over $2. The HMC5883L part is now obsolete, however, and some of the boards claiming to be based on the HMC5883L actually use copies of the part. It took a bit of detective work to get mine reading position since a different I2C I/O assignment was needed compared to the HMC5883L.

My goal is to have a more accurate readout of azimuth and elevation position for my 2M EME system. Anthony Good, K3NG with the help of others, has written Arduino code for a complete az/el control system. Information on the project and the code are available on the web (Reference 2). My particular challenge with this code is figuring out how to configure it since the code is flexible enough to be used with a variety of rotors, displays, position readouts, etc.

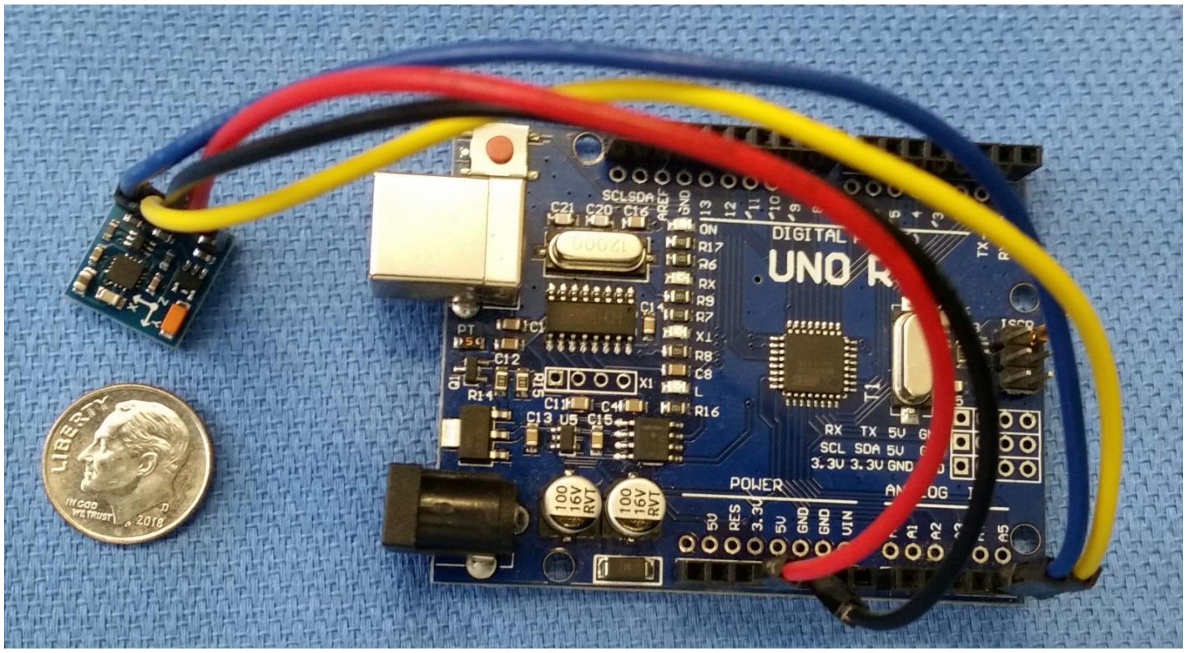

Figure 7 3-axis Magnetic Sensor Interface to an Uno R3

Current study is indicating that the GY-271 makes a good tilt-compensated electronic compass but an accelerometer will also be needed to get both azimuth and elevation readings. One candidate current part to do this is the TDK ICM-20948. It is a 9-degree of freedom (DOF) inertial measurement unit (IMU) that includes a:

- 3-axis gyro

- 3-axis accelerometer
- 3-axis compass
- And, a digital motion processor.

Another similar part is the STMicroelectronics LSM303D. Small boards with these and the supporting parts needed to interface to an Arduino are available for low cost from multiple sources. These parts use micro electromechanical systems (MEMS) technology. These or similar parts are used in smart phones and thus we have relatively cheap access to some powerful sensor technologies that can be exploited with Arduino or other microcontrollers.

These examples hopefully give you an idea of how Arduino or other microcontrollers can be used by the microwave amateur operator. The variety of controllers and add-on hardware available for low-cost is great. Also, the Arduino Integrated Development Environment (IDE) is available free and is a powerful tool for developing the code (sketches) and loading them onto the controllers (Reference 3).

References

1. "Arduino Controlled Microwave Frequency Sources", G. McIntire, AA5C, Microwave Update Proceedings 2017.
2. https://github.com/k3ng/k3ng_rotator_controller
3. https://www.arduino.cc/en/Main/Software