Avocado User Guide

Wildcat Robot Design Studio, Winter/Spring 2018

Akash Borde, Tsong Chen, Rachel Hughes, Weilin Ma, Ryan Miller, Nathan Shelly, Allen Tang

Table of Contents

[User Guide 2](#_Toc516562285)

[At a Glance 2](#_Toc516562286)

[Getting Started 2](#_Toc516562287)

[Operation 7](#_Toc516562288)

[Troubleshooting 9](#_Toc516562289)

[Reference 9](#_Toc516562290)

[API Guide 10](#_Toc516562291)

# User Guide

### At a Glance

Welcome to the Avocado: a lean, inexpensive, high-power density actuator intended for use in closed and open kinematic loops “arms” for walking, hopping, and running research.

#### Features

###### **Cost**

* Low price of $500!

###### Power/Torque

* 30W core power
* 2N-m instantaneous stall torque
* 1N-m stall torque capability for 1-minute duration

###### Mechanical

* Through-hole component for easy wire-harnessing
* Less than 0.5° backlash
* Built-in heatsinking

###### Electrical/Software

* Python and C++ API
* Controls
* Current, position, and velocity control
* User-settable safety thresholds (temperature, current) with emergency stop functionality

### Getting Started

#### Materials

###### In your box

• Avocado

• Power supply (24V, 12.50A) and cables

• Mounting bracket and M4 mounting screws

###### You will need

• Standard wall power outlet (110V 50-60Hz AC)

• Tiva TM4C1294XL microcontroller with LaunchPad by Texas Instruments

• Code Composer Studio software package Texas Instruments

• Jumper cables for serial communication

• Chips and Guacamole (optional)

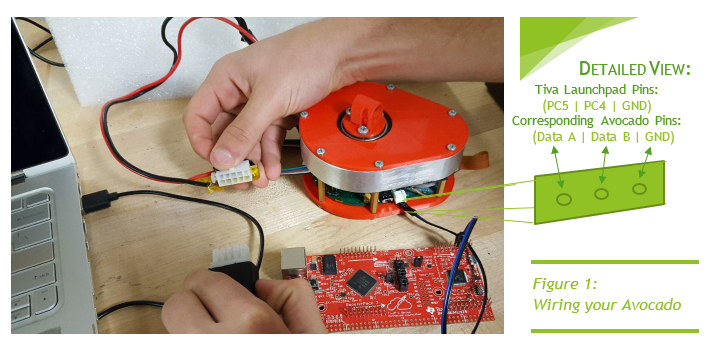
#### Setting up your Avocado

##### Powering Up/Wiring

1. Take the 3-pin Molex communication cable and plug the Molex connector into the 3-pin Molex plug in the Avocado. On the other side, plug the red wire into PC4 of a Tiva TM4C1294XL (or other compatible device) and the green wire into PC5.
2. Plug the Tiva into your computer and open Code Composer Studio.
3. Connect the white power connector into the power supply, and plug the power supply into the wall. If there is a green light on the PCB, you have powered it on successfully! See Figure 1, below, for what it should look like.

CAUTION: Use in the presence of magnetic fields may interfere with the encoder.

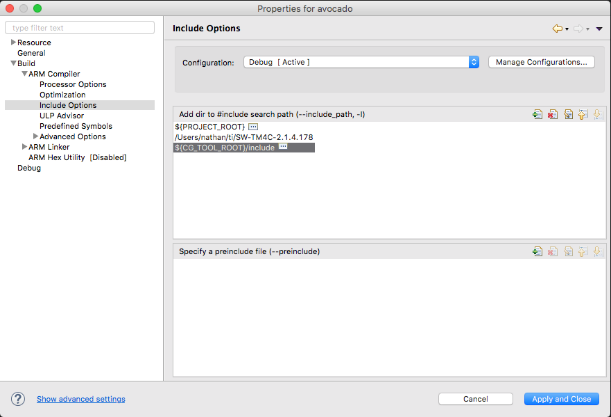
Figure 1: Powered up and wired Avocado



#### Configuring Code Composer Studio

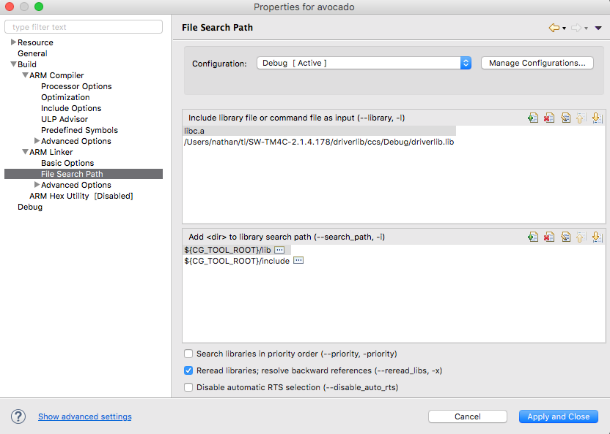
1. In your existing CCS project, we need to set a few CCS project properties:
   * Under Build -> ARM Compiler -> Include Options in the Add dir to #include search path pane add your TivaWara C Series folder (the folder that contains your examples, driverlib, etc.). For me this folder is ~/ti/SW-TM4C-2.1.4.178 (*Note:* the folder name may be different in Windows, an example path is C:\ti\TivaWare\_C\_Series-2.1.4.178)

Figure 2: Building CCS

[](https://raw.githubusercontent.com/Avocado-Actuator/embedded/assets/images/include_options.png)

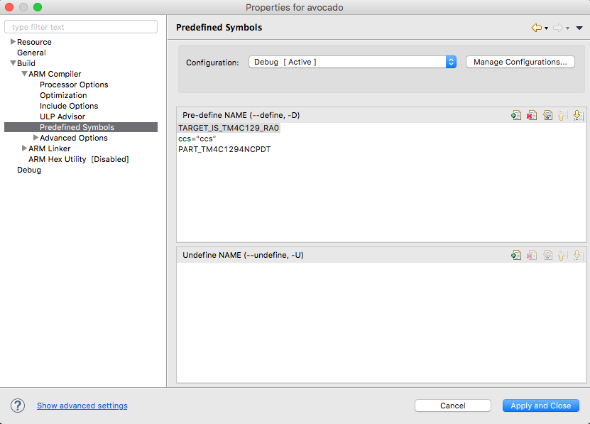
Next under Build -> ARM Linker -> File Search Path in the Include library file or command file as input pane add the full path to your driverlib.lib file. For me this path is ~/ti/SW-TM4C-2.1.4.178/driverlib/ccs/Debug/driverlib.lib, on Windows an example path might be C:\ti\TivaWare\_C\_Series-2.1.4.178\driverlib\ccs\Debug\driverlib.lib.

Figure 3: Locating the library

[](https://raw.githubusercontent.com/Avocado-Actuator/embedded/assets/images/file_search_path.png)

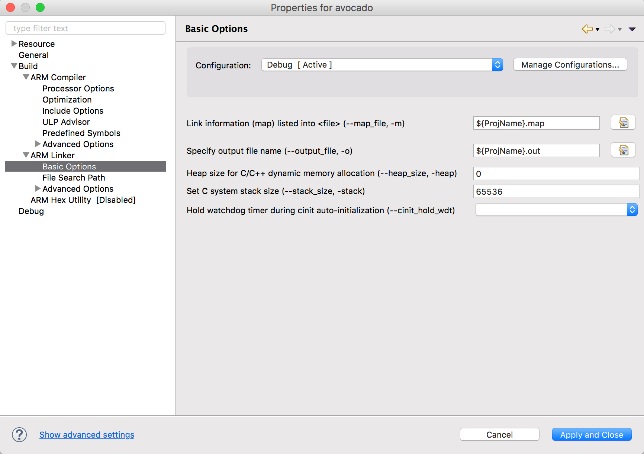
Penultimately, under Build -> ARM Compiler -> Predefined Symbols in the Pre-define NAME pane add TARGET\_IS\_TM4C129\_RA0.

Figure 4: Continued building of CCS

[](https://raw.githubusercontent.com/Avocado-Actuator/embedded/assets/images/predefined_symbols.png)

Finally, under Build -> ARM Linker -> Basic Options set the Set C system stack size option to 65536 .

Figure 5: Setting the stack size

[](https://raw.githubusercontent.com/Avocado-Actuator/embedded/assets/images/stack_size.png)

More in-depth resources for running the Avocado can be found at the git repository [here](https://github.com/Avocado-Actuator).

#### Running Your Program

The c\_library repo holds code for a master controller communicating with avocado actuators. The [export](https://github.com/avocado-actuator/c_library/tree/export) branch contains the library in a pair of header and source files - avocomms.c & avocomms.h. To use this library you should copy these two files to the root of your CCS project. If you'd prefer to write raw messages yourself feel free to examine the protocol [here](https://github.com/Avocado-Actuator/embedded/blob/master/protocol.md).

##### Minimal Usage

Copy over avocomms.c & avocomms.h, then update your ...startup\_ccs.c to enable the interrupts needed by our library. If you have no other interrupts feel free to copy over tm4c1294ncpdt\_startup\_ccs.c to your project, otherwise you'll specifically want to add external declarations for UARTIntHandler, ConsoleIntHandler & Timer0IntHandler (see lines 59-61 of our ...startup\_ccs.c file for a reference). Then make sure to add these interrupts in the appropriate locations in the vector table (again see ...startup\_ccs.c for guidance here).

You should now be able to communicate with your avocado! The first thing you'll want to do is set its address with the function setAddress. Once you've done that you can begin setting position, velocity, current & more.

Here is a minimal main function that moves the avocado to an angle of 60°.

int main(void) {

// set clocking to run directly from the crystal at 120MHz

g\_ui32SysClock = MAP\_SysCtlClockFreqSet((SYSCTL\_XTAL\_25MHZ | SYSCTL\_OSC\_MAIN

| SYSCTL\_USE\_PLL | SYSCTL\_CFG\_VCO\_480), 120000000);

ROM\_IntMasterEnable(); // enable processor interrupts

// initialize communication across UART

CommsInit(g\_ui32SysClock);

// initialize timers for heartbeats

TimerInit(g\_ui32SysClock);

// optional, only if you want debugging readouts

ConsoleInit(); // initialize UART0 for debugging output using UARTStdio

// set avocado's address to 1

setAddress(1);

// rotate avocado with address 1 (what we just set) to 60°

rotateToPosition(1, 60);

}

##### API

For a more detailed look at the API, please see page 10.

#### Video

A video for the Avocado can be found [here](AvocadoVideo.mp4).

### Operation

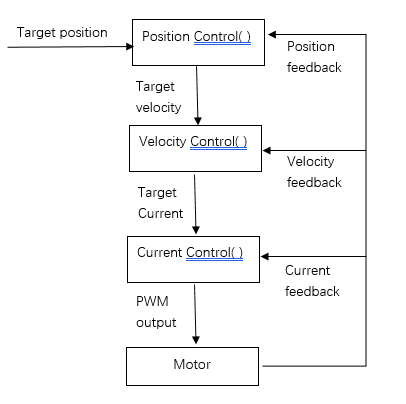
#### Mounting to External Devices

The Avocado can be mounted directly to the output rotor at the top, as shown in Figure 7 of the Reference. The screws required are M4.

#### Control Modes

The Avocado has 3 control modes: torque control, velocity control and position control. They behave according to Figure 6, below. Each control method can be used separately by sending a command through the RS232 communication.

Figure 6: Control Flow



##### Torque

In torque control mode, the MCU will execute CurrentControl() and the motor will run with a constant current. This function takes target\_current as argument and a PID control will adjust the PWM output to control the current in the motor, which is proportional to torque.

To call this function, send command: set cur *val* in protocol, where *val* is the target\_current in units of amp.

##### Position

In position control mode, the MCU executes PositionControl() and the motor will run to a certain position and then hold it there. This function takes target\_position as argument and calculate the proper velocity. Then it will can VelocityControl() recursively.

To call this function, send command: set pos *val* in protocol, where *val* is the target\_position in unit of degree.

##### Velocity

In velocity control mode, the MCU executes VelocityControl() and the motor will run with a constant velocity. This function takes target\_velo as argument and calculate the proper current output. Then it calls CurrentControl() recursively.

To call this function, send command: set vel *val* in protocol, where *val* is the target\_velocity in units of rotations per second.

### Troubleshooting

#### Contact info

With any questions or concerns, contact:

Akash Borde

[Akashborde2019@u.northwestern.edu](mailto:Akashborde2019@u.northwestern.edu)

or

Rachel Hughes

[Rachelhughes2019@u.northwestern.edu](mailto:Rachelhughes2019@u.northwestern.edu)

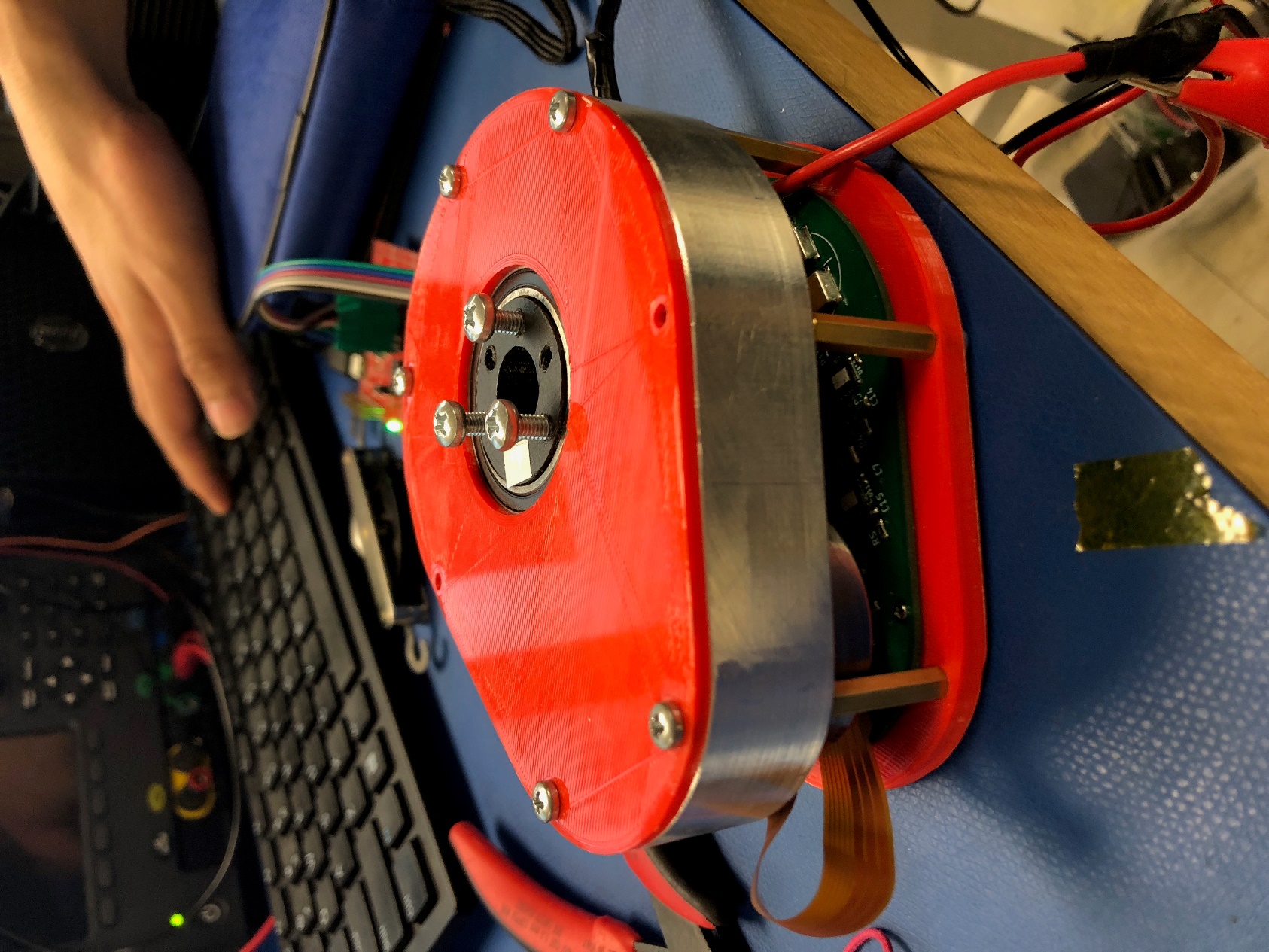
### Reference

#### LegenD

Figure 7, below, shows the primary features of the Avocado.

Figure 7: The Avocado features

*Output Rotor*



*Motor Cable*

# API Guide

The Avocado comes with a control library that you can include in your main control code project. The included C library is optimized to work with Tiva 1294 Microcontrollers.

All of the below functions, except for heartbeat(), return an integer between 0 and 255, inclusive. This number is the message ID of that request. When a request (either a **get** or **set**) generates a response from an Avocado, the response will be stored in the **response\_buffer** struct indexed by the message ID upon arrival. Check **response\_buffer[**messageID**]** to get the result of the response.

The **response\_buffer** struct is made of **Flyte** unions. Access the **f** field at a specific index to read the result if it’s of type float (the result of most **get** requests), and access the **bytes[0]** field if the result is a single byte (from most **set** requests). Note that the message ID wraps around to 0 after it reaches 255, so there is a 255 message history limit.

Below is the list of user-callable functions in the provided control library.

##### getPosition(byte addr)

Gets a four-byte float representing the current angle, in degrees, of the output gear.

##### rotateToPosition(byte addr, float position)

Sets the target position for the Avocado. The Avocado will rotate to the specified angle (in degrees) and hold there.

##### getVelocity(byte addr)

Gets a four-byte float representing the velocity, in rotations per minute, of the output gear.

##### rotateAtVelocity(byte addr, float velocity)

Sets the target velocity for the Avocado. The Avocado will maintain the specified velocity until it either is given a new command or the effort of maintaining the specified velocity under load trips one of the Avocado’s safety thresholds. Passing in a negative velocity will cause the Avocado to rotate in the opposite direction.

##### getCurrent(byte addr)

Gets a four-byte float representing the current, in amps, that the Avocado is drawing.

##### rotateAtCurrent(byte addr, float current)

Sets the target current for the Avocado. The Avocado will maintain the specified current draw until it either is given a new command or the effort of maintaining the specified current under load trips one of the Avocado’s safety thresholds.

##### getMaxCurrent(byte addr)

Gets a four-byte float representing the maximum user-allowed current, in amps, of the Avocado.

##### setMaxCurrent(byte addr, float current)

Sets a maximum current limit for the Avocado. If the Avocado begins to draw more than this value, it will begin to limit its performance to maintain current draw under this value. This value is optional, and if not set, the pre-programmed absolute current limit will take effect.

##### getStopBehavior(byte addr)

Gets a single byte representing the Avocado’s emergency stop behavior. A value of 1 means the Avocado will hold its position as long as no safety thresholds are passed. A value of 0 means the Avocado will kill power to its motor. An emergency stop will happen if the Avocado does not receive heartbeats from the main controller within 500ms intervals.

##### setEStopBehavior(byte addr, byte behavior)

Sets the Avocado’s emergency stop behavior. Using a value of 1 means the Avocado will hold its position as long as no safety thresholds are passed. Using a value of 0 means the Avocado will kill power to its motor. An emergency stop will happen if the Avocado does not receive heartbeats from the main controller within 500ms intervals.

##### getTemperature(byte addr)

Gets a four-byte float representing the temperature of the Avocado in Celsius.

##### getStatus(byte addr)

Gets the Avocado’s status register. For the meaning of the contents, see the section on communication protocol.

##### setAddress(byte addr)

Sets the Avocado’s address. As of project submission, addresses are not persistent through resets. NOTE: This function must only be run when there is one Avocado connected to the controller, or else every Avocado connected along the communication bus will be assigned the specified address. This will produce communication errors and signal overlap.

##### heartbeat()

Sends out a pulse to all Avocados on the communication bus. This is required for Avocado functionality. If the brain shuts off, the communication bus is compromised, or some other drastic event occurs, the Avocados will no longer receive heartbeat pulses and they will initiate their emergency stop behavior. The timeout interval is 500ms.