Arduino program flow should be as follows:

- Prepare for measurement: `PC_CALIBRATION`, `PC_SET_UP_ADCS`, `PC_SET_VOLTAGE` (first voltage to measure, but do not care about data that may come back, after dacSettlingTime has elapsed), wait for `PC_OK` reply (dacSettlingTime is over), then `PC_AUTOGAIN`
- Measure IV curve: `PC_SET_VOLTAGE` (again the same), wait for `PC_OK` reply, trigger camera and wait for data to come back; then `PC_SET_VOLTAGE` (next voltage step), ...

1 Communication with PC

Communication protocol
Each communication message between PC and Arduino is composed of `MSG_START` + 1 byte for length of message that follows, before encoding + `message` + `MSG_END`. Each byte in `message` is encoded as follows: if the ASCII representation of the byte would be larger or equal than a `MSG_SPECIAL_BYTE`, the byte is transformed into 2 bytes, the first one is `MSG_SPECIAL_BYTE`, the second is `byte` − `MSG_SPECIAL_BYTE`.

From PC to Arduino
The Arduino will in some cases expect a second message, including data relative to the original message sent. Here a list of the data expected in the second message, as well as the replies expected for successful completion. Neither 'Data bytes' nor 'Reply bytes' take into account encoding, so the messages may be twice as long. The byte with the message length is also not counted (so 1 more byte).

Original message	Data bytes	Meaning of data bytes	Reply	Reply bytes
`PC_CONFIGURATION`	—	—	version + hardware	4
`PC_CALIBRATION`	3	adcRate, adc0Ch., adc1Ch.	(done) `PC_OK`	1
`PC_SET_UP_ADCS`	4	no. meas. MSB, no. meas. LSB, adc0Ch., adc1Ch.	(data OK) `PC_OK`	1
`PC_AUTOGAIN`	—	—	(done) `PC_OK`	1
`PC_SET_VOLTAGE`	4	dacValue MBS, dacValue LSB, dacSettlingTime MSB, dacSettlingTime LSB	(done waiting) `PC_OK`; (meas. ready) 3×data	1 3×4

From Arduino to PC
The PC can expect to receive:

- 4 bytes containing the information on the current firmware version (first 2 bytes) and hardware settings (last 2), as a reply to `PC_CONFIGURATION`. The firmware version comes as (MAJOR, MINOR), i.e., the maximum is v255.255. The bits in the hardware information have the following meaning:

Bit	Mask	Meaning
0 (`0x01`)	`ADC_0_PRESENT`	(set) ADC#0 is available
1 (`0x02`)	`ADC_1_PRESENT`	(set) ADC#1 is available
2 (`0x04`)	`LM35_PRESENT`	(set) an LM35 temperature sensor is available
3 (`0x08`)	`RELAY_PRESENT`	(set) a relay is available, which allows switching the range of the I_0 measured from the LEED electronics between (0–2.5 V) and (0–10 V)
4 (`0x10`)	`JP_I0_CLOSED`	(set) I_0 input range is 0–2.5 V; (reset) I_0 input range is 0–10 V
5 (`0x20`)	`JP_AUX_CLOSED`	(set) AUX input range is 0–2.5 V; (reset) AUX input range is 0–10 V

Notes: (1) from the point of view of the PC, it probably makes sense to only look at the four least-significant bits, as measurement results are already reported back in physical units (volts, microamperes,

degrees centigrades). (2) When measuring I_0 at the LEED electronics, the return value is in volts. Thus the PC needs to know the V$\leftrightarrow$ μA conversion for the specific setup. (3) While there is an option to use a relay to electronically switch the input range for I_0, this is currently not implemented. Refer to Issue #13 on the viperleed/viperleed Github.

- `PC_OK` in response to `PC_CALIBRATION` once the calibration is over (takes ≈ 3 s to complete).
- `PC_OK` in response to `PC_SET_UP_ADCS`, after the data received by the Arduino (i.e., the message following `PC_SET_UP_ADCS`) has been deemed acceptable and processed. Processing should be relatively fast, as it basically just loads calibration registers into the ADCs, and triggers the conversion of their inputs.
- `PC_OK` in response to `PC_AUTOGAIN`, after the whole gain-finding procedure is over (roughly after 70 ms).
- Three consecutive messages (i.e., each with `MSG_START` and `MSG_END` characters), each with the 4-byte representation of a 32-bit floating-point number. This is the second message after `PC_SET_VOLTAGE`.
- Error messages from several possible states. Each error causes two messages to be sent: The first one is `PC_ERROR` (1 byte), the second one is a 2-byte data message, where the first byte corresponds to the state in which the error occurred, while the second one is one of the error codes in the following Table.

Error code	Value	Reason
`ERROR_SERIAL_OVERFLOW`	1	Arduino serial buffer is full (64 unprocessed characters). The PC sent too many requests that could not be processed in time. The current contents of the serial buffer will be discarded. Wait a few milliseconds before sending a new message to make sure the buffer is fully empty.
`ERROR_MSG_TOO_LONG`	2	The PC sent a message that contained too many characters to be processed. Either the message was corrupted, or the firmware version is incompatible.
`ERROR_MSG_INCONSITENT`	3	The message received is inconsistent. Typically when the number of bytes effectively read does not match the one expected from the info in the message itself. Typically means that the message got corrupted.
`ERROR_MSG_UNKNOWN`	4	The PC sent an unknown command, i.e., it is neither one of those in paragraph *From PC to Arduino* at the beginning of this Section (for 1-byte long messages), or it is a data message ($>$ 1 byte) but we were not waiting for any data to arrive.
`ERROR_MSG_DATA_INVALID`	5	The message was understood, but some of the data passed is inappropriate, or there is too much/little data.
`ERROR_NEVER_CALIBRATED`	6	The ADCs need to be calibrated at least once with `PC_CALIBRATION` after the boot up of the Arduino or after a `PC_RESET`. This has not been done for the channels in use.
`ERROR_TIMEOUT`	7	It was not possible to complete an operation in time. It may mean that (i) the Arduino was waiting for data from the PC that never arrived, or (ii) the internal communication with the ADCs was interrupted, likely because the power supply was disconnected while running.
`ERROR_ADC_SATURATED`	8	The input of one of the ADCs reached solid saturation, and it is not possible to decrease the gain further. May signal an internal malfunction, or that an input cable is incorrectly connected.
`ERROR_TOO_HOT`	9	The temperature measured by the LM35 is very high. There may be some internal malfunction (either the sensor or the board).
`ERROR_RUNTIME`	255	The firmware is corrupt or there is a bug. Some function has been called with inappropriate values or while the Arduino is in the wrong state.

2 State machine

Here all the states of Arduino, including which event triggers entering this state, which function contains the code that is relevant for the state, and whether the state leads to a new state after it is successfully completed.

State	Initiated by	Handler	Goes to state
STATE_GET_CONFIGURATION	PC_CONFIGURATION	getConfiguration()	STATE_IDLE
STATE_CALIBRATE_ADCS	PC_CALIBRATION	calibrateADCsAtAllGains()	STATE_IDLE
STATE_SET_UP_ADCS	PC_SET_UP_ADCS	prepareADCsForMeasurement()	STATE_IDLE
STATE_SET_VOLTAGE	PC_SET_VOLTAGE	setVoltageWaitAndTrigger()	STATE_MEASURE_ADCS
STATE_AUTOGAIN_ADCS	PC_AUTOGAIN	findOptimalADCGains()	STATE_IDLE
STATE_MEASURE_ADCS	STATE_TRIGGER_ADCS	measureADCs()	STATE_ADC_VALUES_READY
STATE_ADC_VALUES_READY	STATE_MEASURE_ADCS	sendMeasuredValues()	STATE_IDLE
STATE_ERROR	Fault	handleErrors()	STATE_IDLE

Follows a description of each of the states in the Arduino finite-state machine:

- STATE_GET_CONFIGURATION: Returns to the PC the current firmware and hardware information. See Section *From Arduino to PC* above for details. Notice that many of the functions require access to the results computed in this state, that should be up to date. Thus, it is advisable to call a PC_CONFIGURATION before each measurement run, or at least after initial boot-up of the Arduino.

- STATE_CALIBRATE_ADCS: In this state the ADCs are self-calibrated in parallel for all the possible gain values. The state expects a data message following PC_CALIBRATION containing 3 bytes: the conversion rate to be used, and the channels of the ADCs to be calibrated. This state takes long to complete (120 ms per calibration point, 3 points per gain to compute medians, 8 gain values. In total ≈ 3 s). Once the calibration is over, it sends back a PC_OK and goes to STATE_IDLE. Many of the states depend on the calibration data gathered in this state. Hence one always needs to make sure that PC_CALIBRATION is issued from the PC. Calibrations expire: drift over time and temperature fluctuations. The PC will need to take care of these things (especially on long-term measurements). All standard IV measurements should include a calibration.

  Possible errors:
  ERROR_TIMEOUT (waiting for data from PC, or no response from ADCs);
  ERROR_MSG_DATA_INVALID (invalid channels or update rate, or less than 3 bytes in the message).

- STATE_SET_UP_ADCS: This state picks the requested channels and fetches pre-stored calibration data. Requires the PC to communicate the number of measurement points that needs to be averaged (2 bytes) as well as which channels are to be measured for the 'external' ADCs (2 bytes). Calls setAllADCgainsAndCalibration(), sends a PC_OK, and returns to STATE_IDLE.

  Possible errors:
  ERROR_TIMEOUT (waiting for data from PC);
  ERROR_MSG_DATA_INVALID (channels are invalid or less than 4 bytes in the message).
  ERROR_NEVER_CALIBRATED (one of the ADC channels is uncalibrated)

- STATE_SET_VOLTAGE: Requires the PC to communicate the DAC value that needs to be set, as well as how long one should wait for the DAC value to be considered stable. It sets the required voltage, and waits dacSettlingTime (milliseconds). After triggering it decreases the ADC gains, should any of the ADCs require this, and goes straight to STATE_MEASURE_ADCS.

  Possible errors:
  ERROR_TIMEOUT (waiting for data from PC).
  ERROR_MSG_DATA_INVALID (not exactly 4 bytes in the message).
  ERROR_NEVER_CALIBRATED (one of the ADC channels is uncalibrated).

- STATE_AUTOGAIN_ADCS: Finds the optimal gain for the available ADCs, measuring 25 values with both ADCs at gain=0 and at 500 Hz. This triggers first a self-calibration for the two ADCs (in parallel),

that takes ≈ 13 ms. Then takes one measurement per state-machine loop, while keeping track of the smallest and largest among the values measured. Finally, chooses the gain such that the worst-case scenario measurement (peak-to-peak, plus the largest among max and min) is above 1/4 of the values that can be measured with that gain. The peak-to-peak measured here is also stored.

Possible errors:
`ERROR_TIMEOUT` (no response from ADC in 5 s)
`ERROR_NEVER_CALIBRATED` (one of the ADC channels is uncalibrated).

- `STATE_MEASURE_ADCS`: Measures the number of values given in `STATE_SET_UP_ADCS` from all the available ADCs, and averages them (one value per state-loop), then goes to `STATE_ADC_VALUES_READY`. During measurement, the values are checked against the saturation thresholds, possibly triggering a gain switch: an immediate gain switch occurs when the value is solidly saturating, if it is possible to reduce the gain; a gain switch is scheduled if the value is not in the central ≈ 50% of the current range. In this case, the actual gain decrease is done the next time `STATE_SET_VOLTAGE` executes.

  NOTES: (1) we are currently throwing away the whole set of measurements if we reach solid saturation. Perhaps we could just `»1` the relevant `summedMeasurements`, and skip the data point for all three ADCs. (2) It may be a good place to check whether the LM35 is reading a temperature that is too high (to discuss what this threshold should be. Probably the `LM35_MAX_ADU` of 80degC is too much).

  Possible errors:
  `ERROR_TIMEOUT` (takes longer than 5 s to measure all the values)
  `ERROR_ADC_SATURATED` (one of the values measured by the external ADCs reached solid saturation, and its gain cannot be decreased further).
  `ERROR_NEVER_CALIBRATED` (one of the ADC channels is uncalibrated).

- `STATE_ADC_VALUES_READY`: This state cannot be reached directly from a PC command. It is the state that automatically follows successful completion of `STATE_MEASURE_ADCS`. Currently it sends back all the data as three separate messages, even if some of the ADCs are not available.