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 ('data' indicates that the reply arrives after the data is received; 'end' means when the operations requested are successfully completed; there always may be an error response, see Section 2). Neither 'Data bytes' nor 'Reply bytes' take into account encoding, so the messages may be twice as long. Also missing is the byte with the message length (so 1 more byte).

Original message	Data bytes	Meaning of data bytes	Reply	Reply bytes
`PC_CONFIGURATION`	—	—	firmware (2 bytes) + hardware (2 bytes)	4
`PC_CALIBRATION`	!—!	—	!—!	—
`PC_SET_UP_ADCS`	5	adc0Channel, adc1Channel, adcRate, no. measurements MSB, no. measurements LSB	data:`PC_OK`	1
`PC_AUTOGAIN`	—	—	end:`PC_OK`	1
`PC_SET_VOLTAGE`	4	dacValue MBS, dacValue LSB, dacSettlingTime MSB, dacSettlingTime LSB	!None! Should do `PC_OK` after dacSettlingTime is elapsed	—
`PC_MEASURE`	—	—	end:3×voltages	3×4

From Arduino to PC
The PC can expect to receive:

- 4 bytes [encoded] 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

  NB: 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).

- WE NEED SOMETHING AS A REPLY TO `PC_CALIBRATION`!

- `PC_OK` [encoded] in response to `PC_SET_UP_ADCS`, after the data received by the Arduino as 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` [encoded] in response to `PC_AUTOGAIN`, after the whole gain-finding procedure is over (roughly after 70 ms).

- Three [encoded] 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.

- Error messages from several possible states. Each error causes two messages to be sent: The first one is the [encoded] `PC_ERROR` byte, the second one is a 2-byte [encoded] data message, where the first byte corresponds to the state in which the error occurred, while the second one is one of these error codes:

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 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.
ERROR_NEVER_CALIBRATED	6	The ADCs need to be calibrated at least once with PC_CALIBRATION after the boot up of the Arduino. This has not been done.
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_SET_UP_ADCS	PC_SET_UP_ADCS	prepareADCsForMeasurement()	STATE_IDLE
STATE_SET_VOLTAGE	PC_SET_VOLTAGE	setVoltage()	STATE_TRIGGER_ADCS
STATE_TRIGGER_ADCS	STATE_SET_VOLTAGE	triggerMeasurements()	STATE_IDLE
STATE_AUTOGAIN_ADCS	PC_AUTOGAIN	findOptimalADCGains()	STATE_IDLE
STATE_MEASURE_ADCS	PC_MEASURE	measureADCs()	STATE_ADC_VALUES_READY
STATE_ADC_VALUES_READY	STATE_MEASURE_ADCS	sendMeasuredValues()	STATE_IDLE
STATE_GET_CONFIGURATION	PC_CONFIGURATION	getConfiguration()	STATE_IDLE
STATE_CALIBRATE_ADCS	PC_CALIBRATION	calibrateADCsAtAllGains()	STATE_IDLE
STATE_ERROR	Fault	handleErrors()	STATE_IDLE

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

- STATE_SET_UP_ADCS: Requires the PC to communicate which channels are to be measured for the external ADCs [currently without bit mask, so LM35 will always be measured!], the measurement frequency (50, 60, or 500 Hz) as well as the number of measurement points that needs to be averaged. Calls setAllADCgainsAndCalibration(), sends an OK to the PC, and returns to STATE_IDLE. In essence, this state just fetches pre-stored calibration data.

  setAllADCgainsAndCalibration() fetches the stored calibration data for the current gain and channel, thus THESE VALUES MUST BE PRESENT AND UP TO DATE. This is currently unchecked, but should be! This means that we must have run through STATE_CALIBRATE_ADCS for the specific channels requested. Thus, we need to somehow keep track of which channels have been calibrated, and which calibrations can still be considered up to date (perhaps we need a long-term timer, and we should also keep track of the temperature [if LM35 is there], since temperature shifts invalidate the self-calibration). It also needs the ADCs to have the correct gains already figured out, so it may be helpful to check whether a STATE_AUTOGAIN_ADCS has successfully run through. Also, it calls AD7705setGainAndTrigger(), i.e. the ADCs start measuring. It is unclear what they are measuring, however, and why, but there should be no communication over the SPI from the ADC back, so it will likely measure a single value and stop there.

  It may reach a STATE_ERROR with (i) ERROR_TIMEOUT if the PC does not send the needed values as a second message, following PC_SET_UP_ADCS, within 5 seconds; (ii) ERROR_REQUEST_INVALID if the PC sent an invalid value for a channel or for the updateRate. Things we may also want to check (and throw a ERROR_REQUEST_PRECEDENCE error otherwise): (a) requires calibration data to be ready (if there are ADCs connected); (b) may give inappropriate results if the gain selected is inappropriate.

- 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 then goes to STATE_TRIGGER_ADCS, after calling setAllADCgainsAndCalibration() should any of the ADCs require its gain to be reduced.

  setAllADCgainsAndCalibration() fetches the stored calibration data for the current gain and channel, thus THESE VALUES MUST BE PRESENT AND UP TO DATE. This is currently unchecked, but should be! It also needs the ADCs to have the correct gains already figured out, so it may be helpful to check whether a STATE_AUTOGAIN_ADCS has successfully run through. In practice, it requires STATE_SET_UP_ADCS to have run through successfully. This is unchecked! Also, it calls AD7705setGainAndTrigger(), i.e. the ADCs start measuring. It is unclear what they are measuring, however, and why, but there should be no communication over the SPI from the ADC back, so it will likely measure a single value and stop there.

  It may reach a STATE_ERROR due to ERROR_TIMEOUT if the PC does not send the needed values as a second message, following PC_SET_VOLTAGE, within 5 seconds. We should probably throw an ERROR_REQUEST_PRECEDENCE if this state is running before a STATE_SET_UP_ADCS has been completed,

i.e., before a PC_SET_UP_ADCS command has been issued. This, however, depends also on which hardware configuration we have (not needed if the ADCs are not there).

- STATE_TRIGGER_ADCS: cannot be induced by PC directly. It is the sub-state that follows successful completion of STATE_SET_VOLTAGE. Does not require any interaction with the PC. For correct operation, requires hardwareDetected to be available and up to date, does nothing if the PC did not ask for PC_CONFIGURATION. Could it do weird stuff if hardwareDetected is not up to date? The Arduino then goes to STATE_IDLE, i.e., although the ADCs have been triggered, NO MEASUREMENT DATA IS SAVED! This means that the PC needs to request measurement explicitly via PC_MEASURE. I don't really see then the use of the automatic switch from STATE_SET_VOLTAGE to STATE_TRIGGER_ADCS, especially if the idea was to have the Arduino take care of the timings to avoid lag from the PC!! IT MUST GO TO STATE_MEASURE_ADCS.

- STATE_AUTOGAIN_ADCS: finds the optimal gain for the available ADCs, measuring 20 values (I THOUGHT WE SAID 25!) with both ADCs at gain=0 and at 500 Hz. This triggers first a self-calibration for the two ADCs (in parallel), that takes $\approx$ 120 ms. Then takes one measurement per state-machine loop (HOW FAST IS A SINGLE MEASUREMENT? COULD WE GET IT FASTER BY TAKING THE 25 MEASUREMENTS WITHOUT GOING THROUGH THE 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. QUESTION: I could not understand whether one needs to explicitly trigger the ADCs before selfCalibrateAllADCs(), or if the stuff done in there via either AD7705selfCalibrate() or AD7705waitForCalibration() already triggers.

  It may reach a STATE_ERROR due to ERROR_TIMEOUT if it takes longer than 5 sec to acquire all the measurements needed.

- STATE_MEASURE_ADCS: Measures a number of values from all the available ADCs, and averages them (one value per state-loop). Requires to know which channels are to be measured, so one should call PC_SET_UP_ADCS beforehand. This is currently unchecked! Currently, it does not tell the PC when the values are ready to be measured, but simply throws them back via the serial line once they are ready, after going to the 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 $\approx$ 50% of the current range, but the actual gain switch currently happens only in STATE_SET_VOLTAGE. This means it does not happen when not measuring a ramp (e.g., a time series at constant energy). Also, we are not checking 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 high). Other observations: (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) The decision on whether we need to switch gain is done based on the currently measured value, but it should actually also take into account of the RIPPLE (which is not set to a value anywhere)!

  It may reach a STATE_ERROR with (i) ERROR_TIMEOUT if more than 5s pass between the receipt of PC_MEASURE and the completion of the nMeasurementsToDo; (ii) ERROR_ADC_SATURATED if one of the values measured by the external ADCs available reaches solid saturation, and its gain cannot be decreased further. Notes on errors that we are not handling: (a) measurement requires the ADCs to be already triggered, so PC_MEASURE cannot be requested before a STATE_TRIGGER_ADCS has been successfully completed. Should throw an ERROR_REQUEST_PRECEDENCE if this is not the case. (b) We may want to also check for possible errors in the LM35, see ERROR_TOO_HOT.

- 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 available data as three separate messages. It would be better to return a single message with the data that was actually requested. Always goes straight to STATE_IDLE at the end and currently cannot go into a STATE_ERROR. We may consider checking whether we come from a STATE_ADC_VALUES_READY, but this may be problematic if we later want to allow the PC to ask for data even if the measurement is not completely over.

- `STATE_GET_CONFIGURATION`: Currently, returns to the PC a 2-byte bit mask indicating the current hardware configuration. Retrieving what is present from the bitmask can be done by bitwise AND with `ADC_0_PRESENT` [bit 0 == LSb], `ADC_1_PRESENT` [bit 1], `LM35_PRESENT` [bit 2], `RELAY_PRESENT` [bit 3], `JP_I0_CLOSED` [bit 4] (I_0 input range is 0–2.5 V instead of 0–10 V), `JP_AUX_CLOSED` [bit 5] (AUX range is 0–2.5 V instead of 0–10 V). For what concerns the PC, only the LSB is likely interesting, since the ADC measurements already return with the correct values. NOTES: (1) when measuring I_0 the return value is in volts. Thus the PC needs to know the V$\leftrightarrow$ μA conversion. (2) I did not see anywhere a way to actually switch the relay electronically, which one may trigger with a command from the PC to switch the input range for the I_0 current. To discuss again if this was meant to be an option. (3) Ideally, we would also like to send back the 2-byte `FIRMWARE_VERSION` before the hardware information (within the same message)

- `STATE_CALIBRATE_ADCS`: In this state the ADCs are self-calibrated in parallel for all the possible gain values and for the currently selected channel. This state take long to complete (120 ms per calibration point, 3 points per gain to compute medians, 8 gain values. In total $\approx$ 3 s). There are several issues with the current implementation. (1) Calibration is always done at 50 Hz. This is because `updateRate` is not available (it currently comes as part of the data message associated with `PC_SET_UP_ADCS`). Solution: have `PC_CALIBRATION` require some data as a second message. One of the bytes is `updateRate`. (2) Calibration is only done for the current channel for both ADCs because the channel information is not available (it currently comes as part of the data message associated with `PC_SET_UP_ADCS`). Solution: have the data message for `PC_CALIBRATION` of the previous point also contain one byte for the channels to be calibrated. This must be done with a bitmask, as we may want to perform a calibration for both channels of the both ADCs. The channels given here may be more than those for which measurements are requested later with `PC_SET_UP_ADCS`. (3) Currently the `initialCalibration()` function cannot time out. This may lead to an unresponsive state in which we wait forever for a self-calibration to finish (e.g., the power plug is disconnected, but the Arduino is still up). (4) once (1+2) are solved, there may be a second way to time out, i.e., no data comes from the PC. (5) Currently the Arduino does not tell back the PC that the calibration requested ran successfully. Thus the PC does not know when it is over. This is problematic, especially considering that this state may last for as long as 6 s.