

## Claw Device Controller

Generated by Doxygen 1.16.0



<b>1 claw</b>	<b>1</b>
<b>2 Data Structure Index</b>	<b>3</b>
2.1 Data Structures	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Data Structure Documentation</b>	<b>7</b>
4.1 stepper_state Struct Reference	7
4.1.1 Detailed Description	7
<b>5 File Documentation</b>	<b>9</b>
5.1 claw.c File Reference	9
5.1.1 Detailed Description	9
5.1.2 Function Documentation	10
5.1.2.1 main()	10
5.2 command_processor.c File Reference	10
5.2.1 Detailed Description	11
5.2.2 Function Documentation	12
5.2.2.1 command_claw_set_position()	12
5.2.2.2 command_get_stepper_status()	12
5.2.2.3 command_move_stepper_absolute()	12
5.2.2.4 command_move_stepper_bump_down()	13
5.2.2.5 command_move_stepper_relative()	13
5.2.2.6 command_move_stepper_rotations()	14
5.2.2.7 command_set_echo()	14
5.2.2.8 command_set_led_period()	14
5.2.2.9 command_set_stepper_period()	15
5.2.2.10 command_set_stepper_zero()	15
5.2.2.11 command_stop_stepper()	15
5.2.2.12 process_command()	16
5.2.2.13 process_stdin_input()	16
5.2.3 Variable Documentation	17
5.2.3.1 help_message	17
5.3 command_processor.h File Reference	17
5.3.1 Detailed Description	18
5.3.2 Function Documentation	18
5.3.2.1 command_claw_set_position()	18
5.3.2.2 command_get_stepper_status()	18
5.3.2.3 command_move_stepper_absolute()	19
5.3.2.4 command_move_stepper_bump_down()	19
5.3.2.5 command_move_stepper_relative()	19
5.3.2.6 command_move_stepper_rotations()	20

5.3.2.7 <code>command_set_echo()</code> . . . . .	20
5.3.2.8 <code>command_set_led_period()</code> . . . . .	21
5.3.2.9 <code>command_set_stepper_period()</code> . . . . .	21
5.3.2.10 <code>command_set_stepper_zero()</code> . . . . .	21
5.3.2.11 <code>command_stop_stepper()</code> . . . . .	22
5.3.2.12 <code>process_command()</code> . . . . .	22
5.3.2.13 <code>process_stdin_input()</code> . . . . .	22
5.4 <code>command_processor.h</code> . . . . .	23
5.5 <code>led.c</code> File Reference . . . . .	23
5.5.1 Detailed Description . . . . .	24
5.5.2 Function Documentation . . . . .	24
5.5.2.1 <code>pico_led_init()</code> . . . . .	24
5.5.2.2 <code>pico_set_led()</code> . . . . .	24
5.5.2.3 <code>process_led_tick()</code> . . . . .	25
5.5.3 Variable Documentation . . . . .	25
5.5.3.1 <code>led_period</code> . . . . .	25
5.6 <code>led.h</code> File Reference . . . . .	25
5.6.1 Detailed Description . . . . .	26
5.6.2 Function Documentation . . . . .	26
5.6.2.1 <code>pico_led_init()</code> . . . . .	26
5.6.2.2 <code>pico_set_led()</code> . . . . .	26
5.6.2.3 <code>process_led_tick()</code> . . . . .	27
5.6.3 Variable Documentation . . . . .	27
5.6.3.1 <code>led_period</code> . . . . .	27
5.7 <code>led.h</code> . . . . .	27
5.8 <code>stepper.c</code> File Reference . . . . .	28
5.8.1 Detailed Description . . . . .	28
5.8.2 Function Documentation . . . . .	28
5.8.2.1 <code>process_stepper_enabled_led()</code> . . . . .	28
5.8.2.2 <code>process_stepper_estop()</code> . . . . .	29
5.8.2.3 <code>process_stepper_movement()</code> . . . . .	29
5.8.2.4 <code>stepper_enable()</code> . . . . .	29
5.8.2.5 <code>stepper_init()</code> . . . . .	30
5.8.2.6 <code>stepper_is_estop_active()</code> . . . . .	30
5.8.2.7 <code>stepper_set_step_period()</code> . . . . .	30
5.8.2.8 <code>stepper_set_target_position()</code> . . . . .	31
5.8.2.9 <code>stepper_stop()</code> . . . . .	31
5.9 <code>stepper.h</code> File Reference . . . . .	31
5.9.1 Detailed Description . . . . .	33
5.9.2 Function Documentation . . . . .	33
5.9.2.1 <code>process_stepper_enabled_led()</code> . . . . .	33
5.9.2.2 <code>process_stepper_estop()</code> . . . . .	33

5.9.2.3 process_stepper_movement()	33
5.9.2.4 stepper_enable()	34
5.9.2.5 stepper_init()	34
5.9.2.6 stepper_is_estop_active()	34
5.9.2.7 stepper_set_step_period()	35
5.9.2.8 stepper_set_target_position()	35
5.9.2.9 stepper_stop()	35
5.10 stepper.h	36
5.11 sys_timer.c File Reference	37
5.11.1 Detailed Description	37
5.11.2 Function Documentation	37
5.11.2.1 timer_callback()	37
5.11.3 Variable Documentation	38
5.11.3.1 ms_ticks_count	38
5.11.3.2 ten_us_ticks_count	38
5.12 sys_timer.h File Reference	38
5.12.1 Detailed Description	39
5.12.2 Function Documentation	39
5.12.2.1 timer_callback()	39
5.12.3 Variable Documentation	39
5.12.3.1 ms_ticks_count	39
5.12.3.2 ten_us_ticks_count	40
5.13 sys_timer.h	40
<b>Index</b>	<b>41</b>



# Chapter 1

## claw

Raspberry pico 2 code to drive stepper motor driven claw





## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">stepper_state</a>	Structure to hold stepper motor state . . . . .	7
-------------------------------	---	---



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">claw.c</a>	Firmware to control a claw device with stepper motors via USB serial commands . . . . .	9
<a href="#">command_processor.c</a>	Implementation of command processing functions . . . . .	10
<a href="#">command_processor.h</a>	Definitions, functions and variables for command processing . . . . .	17
<a href="#">led.c</a>	Implementations and variables for LED control . . . . .	23
<a href="#">led.h</a>	Definitions, function definitions and variables for LED control . . . . .	25
<a href="#">stepper.c</a>	Implementation of stepper motor control and related global variables . . . . .	28
<a href="#">stepper.h</a>	Definitions, functions and variables for stepper motor control . . . . .	31
<a href="#">sys_timer.c</a>	Implementation of system timer and related global variables . . . . .	37
<a href="#">sys_timer.h</a>	Definitions, functions and variables for system timer . . . . .	38



## Chapter 4

# Data Structure Documentation

### 4.1 stepper\_state Struct Reference

Structure to hold stepper motor state.

```
#include <stepper.h>
```

#### Data Fields

- int **current\_position**  
*Current position in steps.*
- int **target\_position**  
*Target position in steps.*
- int **step\_period**  
*Step period in TIMMER\_INTERVAL\_US units.*
- bool **moving**  
*Is the stepper currently moving.*
- bool **enabled**  
*Is the stepper enabled.*

#### 4.1.1 Detailed Description

Structure to hold stepper motor state.

The documentation for this struct was generated from the following file:

- [stepper.h](#)



# Chapter 5

## File Documentation

### 5.1 claw.c File Reference

Firmware to control a claw device with stepper motors via USB serial commands.

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/timer.h"
#include <string.h>
#include "pico/assert.h"
#include <stdlib.h>
#include "hardware/gpio.h"
#include <ctype.h>
#include <math.h>
#include "sys_timer.h"
#include "stepper.h"
#include "led.h"
#include "command_processor.h"
```

#### Functions

- int `main` ()  
*Main function.*

#### 5.1.1 Detailed Description

Firmware to control a claw device with stepper motors via USB serial commands.

##### Author

Jon Wade

##### Date

19 Dec 2025

##### Copyright

(c) 2025 Jon Wade. Standard MIT License applies. See LICENSE file.

Command interface for controlling stepper motors and other functions associated with the claw device. This file implements a simple command interface over USB serial to control the claw device.

## 5.1.2 Function Documentation

### 5.1.2.1 main()

```
int main ()
```

Main function.

#### Parameters

	none
--	------

#### Returns

: none

## 5.2 command\_processor.c File Reference

implementation of command processing functions

```
#include <stdio.h>
#include "pico/stdlib.h"
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include "sys_timer.h"
#include "stepper.h"
#include "led.h"
#include "command_processor.h"
```

#### Macros

- #define **MAX\_COMMAND\_LENGTH** 50
- #define **CLAW\_SET\_POSITION\_COMMAND** "claw\_set "
- #define **LED\_PERIOD\_COMMAND** "led\_period "
- #define **SET\_STEPPER\_PERIOD\_COMMAND** "set\_stepper\_period "
- #define **SET\_STEPPER\_ZERO\_COMMAND** "set\_stepper\_zero"
- #define **MOVE\_STEPPER\_ABSOLUTE\_COMMAND** "move\_stepper\_absolute "
- #define **MOVE\_STEPPER\_RELATIVE\_COMMAND** "move\_stepper\_relative "
- #define **MOVE\_STEPPER\_ROTATIONS\_COMMAND** "move\_stepper\_rotations "
- #define **MOVE\_STEPPER\_BUMP\_DOWN\_COMMAND** "move\_stepper\_bump\_down"
- #define **STOP\_STEPPER\_COMMAND** "stop\_stepper"
- #define **GET\_STEPPER\_STATUS\_COMMAND** "get\_stepper\_status"
- #define **ENABLE\_STEPPER\_COMMAND** "enable\_stepper"
- #define **DISABLE\_STEPPER\_COMMAND** "disable\_stepper"
- #define **ECHO\_COMMAND** "echo "



## Functions

- bool `process_command` (const char \*cmd, stepper\_state\_t \*stepper)  
*Process a command string.*
- char \* `process_stdin_input` (void)  
*Process stdin input.*
- bool `command_get_stepper_status` (stepper\_state\_t \*stepper)  
*Command helper function to get stepper status.*
- bool `command_claw_set_position` (stepper\_state\_t \*stepper, const char \*cmd)  
*Command helper function to set claw position.*
- bool `command_set_led_period` (const char \*cmd)  
*Command helper function to set LED period.*
- bool `command_set_stepper_period` (stepper\_state\_t \*stepper, const char \*cmd)  
*Command helper function to set stepper period.*
- bool `command_set_stepper_zero` (stepper\_state\_t \*stepper)  
*Command helper function to set stepper position to zero.*
- bool `command_move_stepper_absolute` (stepper\_state\_t \*stepper, const char \*cmd)  
*Command helper function to move stepper to an absolute position.*
- bool `command_move_stepper_relative` (stepper\_state\_t \*stepper, const char \*cmd)  
*Command helper function to move stepper by a relative amount.*
- bool `command_move_stepper_rotations` (stepper\_state\_t \*stepper, const char \*cmd)  
*Command helper function to move stepper by a relative amount.*
- bool `command_move_stepper_bump_down` (stepper\_state\_t \*stepper)  
*Command helper function to move stepper by a relative amount.*
- bool `command_stop_stepper` (stepper\_state\_t \*stepper)  
*Command helper function to stop stepper movement.*
- bool `command_set_echo` (const char \*cmd)  
*Command helper function to set command echoing.*

## Variables

- const char \* `help_message`  
*Help message.*
- bool `echo_command` = true

### 5.2.1 Detailed Description

implementation of command processing functions

#### Author

Jon Wade

#### Date

20 Dec 2025

#### Copyright

(c) 2025 Jon Wade. Standard MIT License applies. See LICENSE file.

This file contains the implementation of functions for processing commands.

## 5.2.2 Function Documentation

### 5.2.2.1 `command_claw_set_position()`

```
bool command_claw_set_position (
    stepper_state_t * stepper,
    const char * cmd)
```

Command helper function to set claw position.

#### Note

: Function is not completely safe, assumes valid command string

#### Parameters

<i>stepper</i>	pointer to stepper state structure
<i>cmd</i>	pointer to command string

#### Returns

: true on success, false on failure

### 5.2.2.2 `command_get_stepper_status()`

```
bool command_get_stepper_status (
    stepper_state_t * stepper)
```

Command helper function to get stepper status.

#### Parameters

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

#### Returns

: true on success, false on failure

### 5.2.2.3 `command_move_stepper_absolute()`

```
bool command_move_stepper_absolute (
    stepper_state_t * stepper,
    const char * cmd)
```

Command helper function to move stepper to an absolute position.

#### Note

: Function is not completely safe, assumes valid command string

**Parameters**

<i>stepper</i>	pointer to stepper state structure
<i>cmd</i>	pointer to command string

**Returns**

: true on success, false on failure

**5.2.2.4 command\_move\_stepper\_bump\_down()**

```
bool command_move_stepper_bump_down (  
    stepper_state_t * stepper)
```

Command helper function to move stepper by a relative amount.

**Parameters**

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

**Returns**

: true on success, false on failure

**5.2.2.5 command\_move\_stepper\_relative()**

```
bool command_move_stepper_relative (  
    stepper_state_t * stepper,  
    const char * cmd)
```

Command helper function to move stepper by a relative amount.

**Note**

: Function is not completely safe, assumes valid command string

**Parameters**

<i>stepper</i>	pointer to stepper state structure
<i>cmd</i>	pointer to command string

**Returns**

: true on success, false on failure

### 5.2.2.6 `command_move_stepper_rotations()`

```
bool command_move_stepper_rotations (
    stepper_state_t * stepper,
    const char * cmd)
```

Command helper function to move stepper by a relative amount.

#### Note

: Function is not completely safe, assumes valid command string

#### Parameters

<i>stepper</i>	pointer to stepper state structure
<i>cmd</i>	pointer to command string

#### Returns

: true on success, false on failure

### 5.2.2.7 `command_set_echo()`

```
bool command_set_echo (
    const char * cmd)
```

Command helper function to set command echoing.

This command enables or disables echoing of input characters back to the user.

#### Note

: Function is not completely safe, assumes valid command string

#### Parameters

<i>cmd</i>	pointer to command string
------------	---------------------------

#### Returns

: true on success, false on failure

### 5.2.2.8 `command_set_led_period()`

```
bool command_set_led_period (
    const char * cmd)
```

Command helper function to set LED period.

#### Note

: Function is not completely safe, assumes valid command string

**Parameters**

<i>cmd</i>	pointer to command string
------------	---------------------------

**Returns**

: true on success, false on failure

**5.2.2.9 `command_set_stepper_period()`**

```
bool command_set_stepper_period (  
    stepper_state_t * stepper,  
    const char * cmd)
```

Command helper function to set stepper period.

**Note**

: Function is not completely safe, assumes valid command string

**Parameters**

<i>stepper</i>	pointer to stepper state structure
<i>cmd</i>	pointer to command string

**Returns**

: true on success, false on failure

**5.2.2.10 `command_set_stepper_zero()`**

```
bool command_set_stepper_zero (  
    stepper_state_t * stepper)
```

Command helper function to set stepper position to zero.

**Parameters**

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

**Returns**

: true on success, false on failure

**5.2.2.11 `command_stop_stepper()`**

---

Generated by Doxygen

```
bool command_stop_stepper (  
    stepper_state_t * stepper)
```

**Parameters**

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

**Returns**

: true on success, false on failure

**5.2.2.12 process\_command()**

```
bool process_command (
    const char * cmd,
    stepper_state_t * stepper)
```

Process a command string.

**Note**

: This function parses the command string and calls the appropriate command helper function. It checks for null pointers and command length.

**Parameters**

<i>cmd</i>	pointer to command string
<i>stepper</i>	pointer to stepper state structure

**Returns**

: true on success, false on failure

**5.2.2.13 process\_stdin\_input()**

```
char * process_stdin_input (
    void )
```

Process stdin input.

**Note**

: This function reads characters from stdin, builds commands, and returns complete command strings.

This function has a simple lock to prevent re-entrancy.

**Parameters**

	none
--	------

**Returns**

: non-null pointer to command string when a complete command is received,

## 5.2.3 Variable Documentation

### 5.2.3.1 help\_message

```
const char* help_message
```

Initial value:

```
=
"\n"
"Available commands:\n"
"  claw_set <position>          - Set the claw position 0 to 100\n"
"  led_period <ms>              - Set the LED blink period in milliseconds\n"
"  set_stepper_period <us>      - Set the stepper motor step period in us\n"
"  set_stepper_zero             - Set the current position to zero\n"
"  move_stepper_absolute <steps> - Move the stepper to an absolute position\n"
"  move_stepper_relative <steps> - Move the stepper by a relative number of steps\n"
"  move_stepper_rotations <rotations> - Move the stepper by a number of rotations\n"
"  move_stepper_bump_down       - Move the stepper down by a small fixed amount\n"
"  stop_stepper                 - Stop the stepper motor\n"
"  get_stepper_status           - Get the current status of the stepper motor\n"
"  enable_stepper               - Enable the stepper motor\n"
"  disable_stepper              - Disable the stepper motor\n"
"  echo <on|off>                - Enable or disable command echoing\n"
"  help                         - Show this help message\n"
"-----\n"
```

Help message.

This message is displayed when the user requests help or enters an unknown command.

## 5.3 command\_processor.h File Reference

Definitions, functions and variables for command processing.

```
#include "stepper.h"
```

### Functions

- bool [process\\_command](#) (const char \*cmd, stepper\_state\_t \*stepper)  
*Process a command string.*
- char \* [process\\_stdin\\_input](#) (void)  
*Process stdin input.*
- bool [command\\_claw\\_set\\_position](#) (stepper\_state\_t \*stepper, const char \*cmd)  
*Command helper function to set claw position.*
- bool [command\\_set\\_led\\_period](#) (const char \*cmd)  
*Command helper function to set LED period.*
- bool [command\\_set\\_stepper\\_period](#) (stepper\_state\_t \*stepper, const char \*cmd)  
*Command helper function to set stepper period.*
- bool [command\\_set\\_stepper\\_zero](#) (stepper\_state\_t \*stepper)  
*Command helper function to set stepper position to zero.*
- bool [command\\_move\\_stepper\\_absolute](#) (stepper\_state\_t \*stepper, const char \*cmd)  
*Command helper function to move stepper to an absolute position.*
- bool [command\\_move\\_stepper\\_relative](#) (stepper\_state\_t \*stepper, const char \*cmd)  
*Command helper function to move stepper by a relative amount.*
- bool [command\\_move\\_stepper\\_rotations](#) (stepper\_state\_t \*stepper, const char \*cmd)

- Command helper function to move stepper by a relative amount.*
- bool `command_move_stepper_bump_down` (stepper\_state\_t \*stepper)  
*Command helper function to move stepper by a relative amount.*
- bool `command_stop_stepper` (stepper\_state\_t \*stepper)  
*Command helper function to stop stepper movement.*
- bool `command_get_stepper_status` (stepper\_state\_t \*stepper)  
*Command helper function to get stepper status.*
- bool `command_set_echo` (const char \*cmd)  
*Command helper function to set command echoing.*

### 5.3.1 Detailed Description

Definitions, functions and variables for command processing.

#### Author

Jon Wade

#### Date

20 Dec 2025

#### Copyright

(c) 2025 Jon Wade. Standard MIT License applies. See LICENSE file.

This file contains the definitions, functions and variables for processing commands.

### 5.3.2 Function Documentation

#### 5.3.2.1 `command_claw_set_position()`

```
bool command_claw_set_position (
    stepper_state_t * stepper,
    const char * cmd)
```

Command helper function to set claw position.

#### Note

: Function is not completely safe, assumes valid command string

#### Parameters

<i>stepper</i>	pointer to stepper state structure
<i>cmd</i>	pointer to command string

#### Returns

: true on success, false on failure



**Parameters**

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

**Returns**

: true on success, false on failure

**5.3.2.3 command\_move\_stepper\_absolute()**

```
bool command_move_stepper_absolute (  
    stepper_state_t * stepper,  
    const char * cmd)
```

Command helper function to move stepper to an absolute position.

**Note**

: Function is not completely safe, assumes valid command string

**Parameters**

<i>stepper</i>	pointer to stepper state structure
<i>cmd</i>	pointer to command string

**Returns**

: true on success, false on failure

**5.3.2.4 command\_move\_stepper\_bump\_down()**

```
bool command_move_stepper_bump_down (  
    stepper_state_t * stepper)
```

Command helper function to move stepper by a relative amount.

**Parameters**

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

**Returns**

: true on success, false on failure

**5.3.2.5 command\_move\_stepper\_relative()**

---

Generated by Doxygen

```
bool command_move_stepper_relative (  
    stepper_state_t * stepper,  
    const char * cmd)
```

**Parameters**

<i>stepper</i>	pointer to stepper state structure
<i>cmd</i>	pointer to command string

**Returns**

: true on success, false on failure

**5.3.2.6 command\_move\_stepper\_rotations()**

```
bool command_move_stepper_rotations (  
    stepper_state_t * stepper,  
    const char * cmd)
```

Command helper function to move stepper by a relative amount.

**Note**

: Function is not completely safe, assumes valid command string

**Parameters**

<i>stepper</i>	pointer to stepper state structure
<i>cmd</i>	pointer to command string

**Returns**

: true on success, false on failure

**5.3.2.7 command\_set\_echo()**

```
bool command_set_echo (  
    const char * cmd)
```

Command helper function to set command echoing.

This command enables or disables echoing of input characters back to the user.

**Note**

: Function is not completely safe, assumes valid command string

**Parameters**

<i>cmd</i>	pointer to command string
------------	---------------------------

**Returns**

: true on success, false on failure

### 5.3.2.8 command\_set\_led\_period()

```
bool command_set_led_period (  
    const char * cmd)
```

Command helper function to set LED period.

#### Note

: Function is not completely safe, assumes valid command string

#### Parameters

<i>cmd</i>	pointer to command string
------------	---------------------------

#### Returns

: true on success, false on failure

### 5.3.2.9 command\_set\_stepper\_period()

```
bool command_set_stepper_period (  
    stepper_state_t * stepper,  
    const char * cmd)
```

Command helper function to set stepper period.

#### Note

: Function is not completely safe, assumes valid command string

#### Parameters

<i>stepper</i>	pointer to stepper state structure
<i>cmd</i>	pointer to command string

#### Returns

: true on success, false on failure

### 5.3.2.10 command\_set\_stepper\_zero()

```
bool command_set_stepper_zero (  
    stepper_state_t * stepper)
```

Command helper function to set stepper position to zero.

**Parameters**

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

**Returns**

: true on success, false on failure

**5.3.2.11 command\_stop\_stepper()**

```
bool command_stop_stepper (  
    stepper_state_t * stepper)
```

Command helper function to stop stepper movement.

**Parameters**

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

**Returns**

: true on success, false on failure

**5.3.2.12 process\_command()**

```
bool process_command (  
    const char * cmd,  
    stepper_state_t * stepper)
```

Process a command string.

**Note**

: This function parses the command string and calls the appropriate command helper function. It checks for null pointers and command length.

**Parameters**

<i>cmd</i>	pointer to command string
<i>stepper</i>	pointer to stepper state structure

**Returns**

: true on success, false on failure

**5.3.2.13 process\_stdin\_input()**

```
char * process_stdin_input (  
    void )
```

**Parameters**

	none
--	------

**Returns**

: non-null pointer to command string when a complete command is received,

## 5.4 command\_processor.h

[Go to the documentation of this file.](#)

```

00001
00011
00012 #ifndef COMMAND_PROCESSOR_H
00013 #define COMMAND_PROCESSOR_H
00014
00015 #include "stepper.h"
00016
00027 bool process_command(const char* cmd, stepper_state_t* stepper);
00028
00040 char* process_stdin_input(void);
00041
00051 bool command_claw_set_position(stepper_state_t* stepper, const char* cmd);
00052
00061 bool command_set_led_period(const char* cmd);
00062
00072 bool command_set_stepper_period(stepper_state_t* stepper, const char* cmd);
00073
00080 bool command_set_stepper_zero(stepper_state_t* stepper);
00081
00091 bool command_move_stepper_absolute(stepper_state_t* stepper, const char* cmd);
00092
00102 bool command_move_stepper_relative(stepper_state_t* stepper, const char* cmd);
00103
00113 bool command_move_stepper_rotations(stepper_state_t* stepper, const char* cmd);
00114
00121 bool command_move_stepper_bump_down(stepper_state_t* stepper);
00122
00129 bool command_stop_stepper(stepper_state_t* stepper);
00130
00137 bool command_get_stepper_status(stepper_state_t* stepper);
00138
00149 bool command_set_echo(const char* cmd);
00150
00151 #endif // COMMAND_PROCESSOR_H

```

## 5.5 led.c File Reference

Implementations and variables for LED control.

```

#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "led.h"

```

**Functions**

- int [pico\\_led\\_init](#) (void)  
*Initialise the LED.*
- void [pico\\_set\\_led](#) (bool led\_on)  
*Turn the LED on or off.*
- void [process\\_led\\_tick](#) (void)  
*Process LED timing tick.*

## Variables

- volatile int `led_period` = LED\_DELAY\_MS  
*LED blink period in milliseconds.*

### 5.5.1 Detailed Description

Implementations and variables for LED control.

#### Author

Jon Wade

#### Date

20 Dec 2025

#### Copyright

(c) 2025 Jon Wade. Standard MIT License applies. See LICENSE file.

This file contains the implementations and variables for controlling an LED.

### 5.5.2 Function Documentation

#### 5.5.2.1 pico\_led\_init()

```
int pico_led_init (  
    void )
```

Initialise the LED.

#### Parameters

	none
--	------

#### Returns

: PICO\_OK on success, error code on failure

#### 5.5.2.2 pico\_set\_led()

```
void pico_set_led (  
    bool led_on)
```

Turn the LED on or off.

**Parameters**

<i>led_on</i>	true to turn on, false to turn off
---------------	------------------------------------

**Returns**

: none

**5.5.2.3 process\_led\_tick()**

```
void process_led_tick (  
    void )
```

Process LED timing tick.

**Parameters**

	none
--	------

**Returns**

: none

**5.5.3 Variable Documentation****5.5.3.1 led\_period**

```
volatile int led_period = LED_DELAY_MS
```

LED blink period in milliseconds.

This variable can be modified via command interface to change the LED blink rate.

**5.6 led.h File Reference**

Definitions, function definitions and variables for LED control.

**Macros**

- `#define LED_DELAY_MS 1000`

## Functions

- int `pico_led_init` (void)  
*Initialise the LED.*
- void `pico_set_led` (bool led\_on)  
*Turn the LED on or off.*
- void `process_led_tick` (void)  
*Process LED timing tick.*

## Variables

- volatile int `led_period`  
*LED blink period in milliseconds.*

### 5.6.1 Detailed Description

Definitions, function definitions and variables for LED control.

#### Author

Jon Wade

#### Date

20 Dec 2025

#### Copyright

(c) 2025 Jon Wade. Standard MIT License applies. See LICENSE file.

This file contains the definitions, function definitions and variables for controlling an LED.

### 5.6.2 Function Documentation

#### 5.6.2.1 `pico_led_init()`

```
int pico_led_init (
    void )
```

Initialise the LED.

#### Parameters

	none
--	------

#### Returns

: PICO\_OK on success, error code on failure

#### 5.6.2.2 `pico_set_led()`



**Parameters**

<i>led_on</i>	true to turn on, false to turn off
---------------	------------------------------------

**Returns**

: none

**5.6.2.3 process\_led\_tick()**

```
void process_led_tick (
    void )
```

Process LED timing tick.

**Parameters**

	none
--	------

**Returns**

: none

**5.6.3 Variable Documentation****5.6.3.1 led\_period**

```
volatile int led_period [extern]
```

LED blink period in milliseconds.

This variable can be modified via command interface to change the LED blink rate.

**5.7 led.h**

[Go to the documentation of this file.](#)

```
00001
00011
00012 #ifndef LED_H
00013 #define LED_H
00014
00015 // Define default LED delay if not defined
00016 #ifndef LED_DELAY_MS
00017 #define LED_DELAY_MS 1000
00018 #endif
00019
00025 extern volatile int led_period;
00026
00033 int pico_led_init(void);
00034
00041 void pico_set_led(bool led_on);
00042
00043
00044
00051 void process_led_tick(void);
00052
00053 #endif
```

## 5.8 stepper.c File Reference

implementation of stepper motor control and related global variables

```
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "stepper.h"
#include "sys_timer.h"
```

### Functions

- bool [stepper\\_init](#) (stepper\_state\_t \*stepper, int initial\_position, int step\_period)  
*Initialize the stepper state.*
- bool [stepper\\_set\\_target\\_position](#) (stepper\_state\_t \*stepper, int target\_position)  
*Set the target position for the stepper motor.*
- bool [stepper\\_set\\_step\\_period](#) (stepper\_state\_t \*stepper, int step\_period\_us)  
*Set the step period for the stepper motor.*
- bool [stepper\\_stop](#) (stepper\_state\_t \*stepper)  
*Stop the stepper motor, setting target position to current position.*
- bool [stepper\\_enable](#) (stepper\_state\_t \*stepper, bool enable)  
*Enable the stepper motor.*
- bool [stepper\\_is\\_estop\\_active](#) (stepper\_state\_t \*stepper)  
*Check if the estop is active and set estop status LED appropriately.*
- bool [process\\_stepper\\_estop](#) (stepper\_state\_t \*stepper)  
*Process stepper estop input.*
- bool [process\\_stepper\\_enabled\\_led](#) (stepper\_state\_t \*stepper)  
*Process stepper enabled LED.*
- bool [process\\_stepper\\_movement](#) (stepper\_state\_t \*stepper)  
*Process stepper movement.*

### 5.8.1 Detailed Description

implementation of stepper motor control and related global variables

#### Author

Jon Wade

#### Date

20 Dec 2025

#### Copyright

(c) 2025 Jon Wade. Standard MIT License applies. See LICENSE file.

This file contains the implementation of functions for controlling a stepper motor.

## 5.8.2 Function Documentation

### 5.8.2.1 process\_stepper\_enabled\_led()

**Parameters**

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

**Returns**

: true on success, false on failure

**5.8.2.2 process\_stepper\_estop()**

```
bool process_stepper_estop (  
    stepper_state_t * stepper)
```

Process stepper estop input.

**Parameters**

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

**Returns**

: true if estop is active, false otherwise

**5.8.2.3 process\_stepper\_movement()**

```
bool process_stepper_movement (  
    stepper_state_t * stepper)
```

Process stepper movement.

**Parameters**

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

**Returns**

: true if stepper is still moving, false if it has reached target

**5.8.2.4 stepper\_enable()**

```
bool stepper_enable (  
    stepper_state_t * stepper,  
    bool enable)
```

Enable the stepper motor.

**Parameters**

<i>stepper</i>	pointer to stepper state structure, must not be NULL
<i>enable</i>	true to enable, false to disable

**Returns**

: true on success, false on failure

**5.8.2.5 stepper\_init()**

```
bool stepper_init (
    stepper_state_t * stepper,
    int initial_position,
    int step_period)
```

Initialize the stepper state.

**Parameters**

<i>stepper</i>	pointer to stepper state structure to initialize, must not be NULL
<i>initial_position</i>	initial position in steps must be between MIN_STEPPER_POSITION and MAX_STEPPER_POSITION
<i>step_period</i>	step period in TIMER_INTERVAL_US must be greater than 1 ms

**Returns**

: true on success, false on failure

**5.8.2.6 stepper\_is\_estop\_active()**

```
bool stepper_is_estop_active (
    stepper_state_t * stepper)
```

Check if the estop is active and set estop status LED appropriately.

**Parameters**

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

**Returns**

: true if estop is active, false otherwise

**5.8.2.7 stepper\_set\_step\_period()**

```
bool stepper_set_step_period (
    stepper_state_t * stepper,
    int step_period_us)
```

**Parameters**

<i>stepper</i>	pointer to stepper state structure, must not be NULL
<i>step_period</i>	step period in microseconds must be greater than MIN_STEPPER_PERIOD

**Returns**

: true on success, false on failure

**5.8.2.8 stepper\_set\_target\_position()**

```
bool stepper_set_target_position (
    stepper_state_t * stepper,
    int target_position)
```

Set the target position for the stepper motor.

**Parameters**

<i>stepper</i>	pointer to stepper state structure, must not be NULL
<i>target_position</i>	target position in steps must be between MIN_STEPPER_POSITION and MAX_STEPPER_POSITION

**Returns**

: true on success, false on failure

**5.8.2.9 stepper\_stop()**

```
bool stepper_stop (
    stepper_state_t * stepper)
```

Stop the stepper motor, setting target position to current position.

**Parameters**

<i>stepper</i>	pointer to stepper state structure, must not be NULL
----------------	--

**Returns**

: true on success, false on failure

**5.9 stepper.h File Reference**

Definitions, functions and variables for stepper motor control.

## Data Structures

- struct [stepper\\_state](#)  
*Structure to hold stepper motor state.*

## Macros

- `#define DEFAULT_STEPPER_PERIOD 4`
- `#define MIN_STEPPER_PERIOD 4`
- `#define STEPPER_STEP_PIN 6`
- `#define STEPPER_DIR_PIN 7`
- `#define STEPPER_ENABLE_PIN 8`
- `#define STEPPER_ENABLE_PIN_INVERTED true`
- `#define STEPPER_ENABLE_LED_PIN 14`
- `#define STEPPER_ENABLE_LED_PIN_ACTIVE_LEVEL 1`
- `#define STEPPER_ESTOP_LED_PIN 15`
- `#define STEPPER_ESTOP_LED_PIN_ACTIVE_LEVEL 1`
- `#define STEPPER_ESTOP_PIN 16`
- `#define STEPPER_ESTOP_ACTIVE_LEVEL 0`
- `#define STEPPER_ESTOP_DEACTIVATE_DELAY_MS 100`
- `#define STEPPER_DIRECTION_FORWARD 1`
- `#define STEPPER_DIRECTION_BACKWARD 0`
- `#define STEPPER_STEPS_PER_REV 3200`
- `#define STEPPER_MAX_REVOLUTIONS 12`
- `#define STEPPER_BUMP_STEPS (STEPPER_STEPS_PER_REV / 4)`
- `#define MAX_STEPPER_POSITION (STEPPER_STEPS_PER_REV * STEPPER_MAX_REVOLUTIONS)`
- `#define MIN_STEPPER_POSITION 0`
- `#define STATUS_LED_ON 1`
- `#define STATUS_LED_OFF 0`

## Typedefs

- typedef struct [stepper\\_state](#) [stepper\\_state\\_t](#)

## Functions

- bool [stepper\\_init](#) ([stepper\\_state\\_t](#) \*stepper, int initial\_position, int step\_period)  
*Initialize the stepper state.*
- bool [stepper\\_set\\_target\\_position](#) ([stepper\\_state\\_t](#) \*stepper, int target\_position)  
*Set the target position for the stepper motor.*
- bool [stepper\\_set\\_step\\_period](#) ([stepper\\_state\\_t](#) \*stepper, int step\_period\_us)  
*Set the step period for the stepper motor.*
- bool [stepper\\_stop](#) ([stepper\\_state\\_t](#) \*stepper)  
*Stop the stepper motor, setting target position to current position.*
- bool [stepper\\_enable](#) ([stepper\\_state\\_t](#) \*stepper, bool enable)  
*Enable the stepper motor.*
- bool [process\\_stepper\\_movement](#) ([stepper\\_state\\_t](#) \*stepper)  
*Process stepper movement.*
- bool [process\\_stepper\\_estop](#) ([stepper\\_state\\_t](#) \*stepper)  
*Process stepper estop input.*
- bool [stepper\\_is\\_estop\\_active](#) ([stepper\\_state\\_t](#) \*stepper)  
*Check if the estop is active and set estop status LED appropriately.*
- bool [process\\_stepper\\_enabled\\_led](#) ([stepper\\_state\\_t](#) \*stepper)  
*Process stepper enabled LED.*

## 5.9.1 Detailed Description

Definitions, functions and variables for stepper motor control.

### Author

Jon Wade

### Date

20 Dec 2025

### Copyright

(c) 2025 Jon Wade. Standard MIT License applies. See LICENSE file.

This file contains the definitions, functions and variables for controlling a stepper motor.

## 5.9.2 Function Documentation

### 5.9.2.1 process\_stepper\_enabled\_led()

```
bool process_stepper_enabled_led (  
    stepper_state_t * stepper)
```

Process stepper enabled LED.

#### Parameters

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

#### Returns

: true on success, false on failure

### 5.9.2.2 process\_stepper\_estop()

```
bool process_stepper_estop (  
    stepper_state_t * stepper)
```

Process stepper estop input.

#### Parameters

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

---

#### Returns

: true if estop is active, false otherwise

**Parameters**

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

**Returns**

: true if stepper is still moving, false if it has reached target

**5.9.2.4 stepper\_enable()**

```
bool stepper_enable (
    stepper_state_t * stepper,
    bool enable)
```

Enable the stepper motor.

**Parameters**

<i>stepper</i>	pointer to stepper state structure, must not be NULL
<i>enable</i>	true to enable, false to disable

**Returns**

: true on success, false on failure

**5.9.2.5 stepper\_init()**

```
bool stepper_init (
    stepper_state_t * stepper,
    int initial_position,
    int step_period)
```

Initialize the stepper state.

**Parameters**

<i>stepper</i>	pointer to stepper state structure to initialize, must not be NULL
<i>initial_position</i>	initial position in steps must be between MIN_STEPPER_POSITION and MAX_STEPPER_POSITION
<i>step_period</i>	step period in TIMER_INTERVAL_US must be greater than 1 ms

**Returns**

: true on success, false on failure

**5.9.2.6 stepper\_is\_estop\_active()**

```
bool stepper_is_estop_active (
    stepper_state_t * stepper)
```



**Parameters**

<i>stepper</i>	pointer to stepper state structure
----------------	------------------------------------

**Returns**

: true if estop is active, false otherwise

**5.9.2.7 stepper\_set\_step\_period()**

```
bool stepper_set_step_period (
    stepper_state_t * stepper,
    int step_period_us)
```

Set the step period for the stepper motor.

**Parameters**

<i>stepper</i>	pointer to stepper state structure, must not be NULL
<i>step_period</i>	step period in microseconds must be greater than MIN_STEPPER_PERIOD

**Returns**

: true on success, false on failure

**5.9.2.8 stepper\_set\_target\_position()**

```
bool stepper_set_target_position (
    stepper_state_t * stepper,
    int target_position)
```

Set the target position for the stepper motor.

**Parameters**

<i>stepper</i>	pointer to stepper state structure, must not be NULL
<i>target_position</i>	target position in steps must be between MIN_STEPPER_POSITION and MAX_STEPPER_POSITION

**Returns**

: true on success, false on failure

**5.9.2.9 stepper\_stop()**

```
bool stepper_stop (
    stepper_state_t * stepper)
```

Stop the stepper motor, setting target position to current position.

**Parameters**

<i>stepper</i>	pointer to stepper state structure, must not be NULL
----------------	--

**Returns**

: true on success, false on failure

**5.10 stepper.h**

[Go to the documentation of this file.](#)

```

00001
00011
00012 #ifndef STEPPER_H
00013 #define STEPPER_H
00014
00015 // Stepper motor configuration
00016 #define DEFAULT_STEPPER_PERIOD          4
00017 #define MIN_STEPPER_PERIOD              4
00018 #define STEPPER_STEP_PIN                6
00019 #define STEPPER_DIR_PIN                 7
00020 #define STEPPER_ENABLE_PIN              8
00021 #define STEPPER_ENABLE_PIN_INVERTED    true
00022 #define STEPPER_ENABLE_LED_PIN         14
00023 #define STEPPER_ENABLE_LED_PIN_ACTIVE_LEVEL 1
00024 #define STEPPER_ESTOP_LED_PIN          15
00025 #define STEPPER_ESTOP_LED_PIN_ACTIVE_LEVEL 1
00026 #define STEPPER_ESTOP_PIN              16
00027 #define STEPPER_ESTOP_ACTIVE_LEVEL     0
00028 #define STEPPER_ESTOP_DEACTIVATE_DELAY_MS 100
00029 #define STEPPER_DIRECTION_FORWARD      1
00030 #define STEPPER_DIRECTION_BACKWARD     0
00031 #define STEPPER_STEPS_PER_REV          3200
00033 #define STEPPER_MAX_REVOLUTIONS        12
00035 #define STEPPER_BUMP_STEPS              (STEPPER_STEPS_PER_REV / 4) // Number of steps to move for
    a bump down command (1/4 revolution)
00036 #define MAX_STEPPER_POSITION            (STEPPER_STEPS_PER_REV * STEPPER_MAX_REVOLUTIONS)
00037 #define MIN_STEPPER_POSITION            0
00038
00039 #define STATUS_LED_ON                   1
00040 #define STATUS_LED_OFF                  0
00041
00045 typedef struct stepper_state
00046 {
00047     int current_position;
00048     int target_position;
00049     int step_period;
00050     bool moving;
00051     bool enabled;
00052 } stepper_state_t;
00053
00054 // Function prototypes
00055
00064 bool stepper_init(stepper_state_t* stepper, int initial_position, int step_period);
00065
00073 bool stepper_set_target_position(stepper_state_t* stepper, int target_position);
00074
00082 bool stepper_set_step_period(stepper_state_t* stepper, int step_period_us);
00083
00090 bool stepper_stop(stepper_state_t* stepper);
00091
00099 bool stepper_enable(stepper_state_t* stepper, bool enable);
00100
00107 bool process_stepper_movement(stepper_state_t* stepper);
00108
00114
00115 bool process_stepper_estop(stepper_state_t* stepper);
00116
00122 bool stepper_is_estop_active(stepper_state_t* stepper);
00123
00124
00130 bool process_stepper_enabled_led(stepper_state_t* stepper);
00131
00132 #endif // STEPPER_H

```

## 5.11 sys\_timer.c File Reference

implementation of system timer and related global variables

```
#include "pico/stdlib.h"
#include "hardware/timer.h"
#include "sys_timer.h"
```

### Functions

- bool [timer\\_callback](#) (struct repeating\_timer \*t)  
*Millisecond timer callback.*

### Variables

- volatile int [ten\\_us\\_ticks\\_count](#) = 0  
*Global ten microsecond ticks count.*
- volatile int [ms\\_ticks\\_count](#) = 0  
*Global millisecond ticks count.*

### 5.11.1 Detailed Description

implementation of system timer and related global variables

#### Author

Jon Wade

#### Date

20 Dec 2025

#### Copyright

(c) 2025 Jon Wade. Standard MIT License applies. See LICENSE file.

This file contains the timer callback function and related definitions for the system timer.

### 5.11.2 Function Documentation

#### 5.11.2.1 timer\_callback()

```
bool timer_callback (  
    struct repeating_timer * t)
```

*Millisecond timer callback.*

---

Generated by Doxygen

#### Note

: This function is called every millisecond by the repeating timer.

**Parameters**

<i>t</i>	pointer to repeating_timer struct
----------	-----------------------------------

**Returns**

: true to keep repeating, false to stop

**5.11.3 Variable Documentation****5.11.3.1 ms\_ticks\_count**

```
volatile int ms_ticks_count = 0
```

Global millisecond ticks count.

This variable is incremented by the timer callback every 100 calls (1 ms = 100 \* 10 us) and decremented in the main loop to track when the millisecond tasks should run.

**5.11.3.2 ten\_us\_ticks\_count**

```
volatile int ten_us_ticks_count = 0
```

Global ten microsecond ticks count.

This variable is incremented by the timer callback and decremented in the main loop to track when the ten microsecond tasks should run.

**5.12 sys\_timer.h File Reference**

Definitions, functions and variables for system timer.

**Macros**

- #define **TIMER\_INTERVAL\_US** 10

**Functions**

- bool [timer\\_callback](#) (struct repeating\_timer \*t)  
*Millisecond timer callback.*

**Variables**

- volatile int [ten\\_us\\_ticks\\_count](#)  
*Global ten microsecond ticks count.*
- volatile int [ms\\_ticks\\_count](#)  
*Global millisecond ticks count.*

## 5.12.1 Detailed Description

Definitions, functions and variables for system timer.

### Author

Jon Wade

### Date

20 Dec 2025

### Copyright

(c) 2025 Jon Wade. Standard MIT License applies. See LICENSE file.

This file contains the timer callback function and related definitions for the system timer.

## 5.12.2 Function Documentation

### 5.12.2.1 timer\_callback()

```
bool timer_callback (  
    struct repeating_timer * t)
```

Millisecond timer callback.

### Note

: This function is called every millisecond by the repeating timer.

### Parameters

<i>t</i>	pointer to repeating_timer struct
----------	-----------------------------------

### Returns

: true to keep repeating, false to stop

## 5.12.3 Variable Documentation

### 5.12.3.1 ms\_ticks\_count

```
volatile int ms_ticks_count [extern]
```

Global millisecond ticks count.

This variable is incremented by the timer callback every 100 calls (1 ms = 100 \* 10 us) and decremented in the main loop to track when the millisecond tasks should run.

### 5.12.3.2 ten\_us\_ticks\_count

```
volatile int ten_us_ticks_count [extern]
```

Global ten microsecond ticks count.

This variable is incremented by the timer callback and decremented in the main loop to track when the ten microsecond tasks should run.

## 5.13 sys\_timer.h

[Go to the documentation of this file.](#)

```
00001
00011
00012 #ifndef SYS_TIMER_H
00013 #define SYS_TIMER_H
00014
00015 #define TIMER_INTERVAL_US          10          // Timer interval in microseconds
00016
00023 extern volatile int ten_us_ticks_count;
00024
00031
00032 extern volatile int ms_ticks_count;
00033
00042 bool timer_callback(struct repeating_timer *t);
00043
00044 #endif // SYS_TIMER_H
```

# Index

- claw, [1](#)
- claw.c, [9](#)
  - main, [10](#)
- command\_claw\_set\_position
  - command\_processor.c, [12](#)
  - command\_processor.h, [18](#)
- command\_get\_stepper\_status
  - command\_processor.c, [12](#)
  - command\_processor.h, [18](#)
- command\_move\_stepper\_absolute
  - command\_processor.c, [12](#)
  - command\_processor.h, [19](#)
- command\_move\_stepper\_bump\_down
  - command\_processor.c, [13](#)
  - command\_processor.h, [19](#)
- command\_move\_stepper\_relative
  - command\_processor.c, [13](#)
  - command\_processor.h, [19](#)
- command\_move\_stepper\_rotations
  - command\_processor.c, [13](#)
  - command\_processor.h, [20](#)
- command\_processor.c, [10](#)
  - command\_claw\_set\_position, [12](#)
  - command\_get\_stepper\_status, [12](#)
  - command\_move\_stepper\_absolute, [12](#)
  - command\_move\_stepper\_bump\_down, [13](#)
  - command\_move\_stepper\_relative, [13](#)
  - command\_move\_stepper\_rotations, [13](#)
  - command\_set\_echo, [14](#)
  - command\_set\_led\_period, [14](#)
  - command\_set\_stepper\_period, [15](#)
  - command\_set\_stepper\_zero, [15](#)
  - command\_stop\_stepper, [15](#)
  - help\_message, [17](#)
  - process\_command, [16](#)
  - process\_stdin\_input, [16](#)
- command\_processor.h, [17](#)
  - command\_claw\_set\_position, [18](#)
  - command\_get\_stepper\_status, [18](#)
  - command\_move\_stepper\_absolute, [19](#)
  - command\_move\_stepper\_bump\_down, [19](#)
  - command\_move\_stepper\_relative, [19](#)
  - command\_move\_stepper\_rotations, [20](#)
  - command\_set\_echo, [20](#)
  - command\_set\_led\_period, [20](#)
  - command\_set\_stepper\_period, [21](#)
  - command\_set\_stepper\_zero, [21](#)
  - command\_stop\_stepper, [22](#)
  - process\_command, [22](#)
  - process\_stdin\_input, [22](#)
  - command\_set\_echo
    - command\_processor.c, [14](#)
    - command\_processor.h, [20](#)
  - command\_set\_led\_period
    - command\_processor.c, [14](#)
    - command\_processor.h, [20](#)
  - command\_set\_stepper\_period
    - command\_processor.c, [15](#)
    - command\_processor.h, [21](#)
  - command\_set\_stepper\_zero
    - command\_processor.c, [15](#)
    - command\_processor.h, [21](#)
  - command\_stop\_stepper
    - command\_processor.c, [15](#)
    - command\_processor.h, [22](#)
- help\_message
  - command\_processor.c, [17](#)
- led.c, [23](#)
  - led\_period, [25](#)
  - pico\_led\_init, [24](#)
  - pico\_set\_led, [24](#)
  - process\_led\_tick, [25](#)
- led.h, [25](#)
  - led\_period, [27](#)
  - pico\_led\_init, [26](#)
  - pico\_set\_led, [26](#)
  - process\_led\_tick, [27](#)
- led\_period
  - led.c, [25](#)
  - led.h, [27](#)
- main
  - claw.c, [10](#)
- ms\_ticks\_count
  - sys\_timer.c, [38](#)
  - sys\_timer.h, [39](#)
- pico\_led\_init
  - led.c, [24](#)
  - led.h, [26](#)
- pico\_set\_led
  - led.c, [24](#)
  - led.h, [26](#)
- process\_command
  - command\_processor.c, [16](#)
  - command\_processor.h, [22](#)
- process\_led\_tick

- led.c, [25](#)
- led.h, [27](#)
- process\_stdin\_input
  - command\_processor.c, [16](#)
  - command\_processor.h, [22](#)
- process\_stepper\_enabled\_led
  - stepper.c, [28](#)
  - stepper.h, [33](#)
- process\_stepper\_estop
  - stepper.c, [29](#)
  - stepper.h, [33](#)
- process\_stepper\_movement
  - stepper.c, [29](#)
  - stepper.h, [33](#)
- stepper.c, [28](#)
  - process\_stepper\_enabled\_led, [28](#)
  - process\_stepper\_estop, [29](#)
  - process\_stepper\_movement, [29](#)
  - stepper\_enable, [29](#)
  - stepper\_init, [30](#)
  - stepper\_is\_estop\_active, [30](#)
  - stepper\_set\_step\_period, [30](#)
  - stepper\_set\_target\_position, [31](#)
  - stepper\_stop, [31](#)
- stepper.h, [31](#)
  - process\_stepper\_enabled\_led, [33](#)
  - process\_stepper\_estop, [33](#)
  - process\_stepper\_movement, [33](#)
  - stepper\_enable, [34](#)
  - stepper\_init, [34](#)
  - stepper\_is\_estop\_active, [34](#)
  - stepper\_set\_step\_period, [35](#)
  - stepper\_set\_target\_position, [35](#)
  - stepper\_stop, [35](#)
- stepper\_enable
  - stepper.c, [29](#)
  - stepper.h, [34](#)
- stepper\_init
  - stepper.c, [30](#)
  - stepper.h, [34](#)
- stepper\_is\_estop\_active
  - stepper.c, [30](#)
  - stepper.h, [34](#)
- stepper\_set\_step\_period
  - stepper.c, [30](#)
  - stepper.h, [35](#)
- stepper\_set\_target\_position
  - stepper.c, [31](#)
  - stepper.h, [35](#)
- stepper\_state, [7](#)
- stepper\_stop
  - stepper.c, [31](#)
  - stepper.h, [35](#)
- sys\_timer.c, [37](#)
  - ms\_ticks\_count, [38](#)
  - ten\_us\_ticks\_count, [38](#)
  - timer\_callback, [37](#)
- sys\_timer.h, [38](#)
  - ms\_ticks\_count, [39](#)
  - ten\_us\_ticks\_count, [39](#)
  - timer\_callback, [39](#)
- ten\_us\_ticks\_count
  - sys\_timer.c, [38](#)
  - sys\_timer.h, [39](#)
- timer\_callback
  - sys\_timer.c, [37](#)
  - sys\_timer.h, [39](#)