

# Assessment 2 component: Camera controller

## Summary

Each camera controller is responsible for a single infrared security camera. These low-resolution cameras are placed throughout the building for security purposes; the overseer can send them commands (such as turn on, turn off, change angle, request frame) and the camera will notify the overseer automatically if any motion is detected. All of these communications happen over TCP.

The cameras capture a grainy 48x36 image, and the infrared characteristics of the building vary, so some processing work needs to be done to determine if relevant motion has taken place or not.

## Program name

camera

## Command-line arguments

```
{id} {address:port} {temperature change threshold} {shared memory path} {shared memory offset} {overseer address:port}
```

## Shared memory structure

```
struct {
    uint16_t current_angle;
    uint16_t min_angle;
    uint16_t max_angle;
    char status; // 'L' for turning left, 'R' for turning right, 'O' for on (and stationary), '-' for off
    uint8_t video[36][48];
    pthread_mutex_t mutex;
    pthread_cond_t cond;
};
```

The camera controller is only permitted to change 'status'. 'current\_angle' is controlled by the camera itself, 'video' is provided by the camera, and 'min\_angle'/'max\_angle' are fixed values set by the camera.

## Initialisation

On startup, this component will bind the port it has been supplied with. It should be non-blocking - use `fcntl(fd, F_SETFL, fcntl(fd, F_GETFL) | O_NONBLOCK);` after opening the socket so that `accept()` does not block. The component will start listening on that socket, then send the following initialisation message to the overseer via TCP:

```
CAMERA {id} {address:port} HELLO#
```

## Commands from overseer

The overseer can, at any point, connect to the camera controller via TCP and send one of the following messages:

- `ON#` - If the camera is currently off, it will turn on
- `OFF#` - If the camera is currently on (whether turning or stationary) it will turn off
- `TURN {angle}#` - Given an (integer) angle between 0 and 359, the camera will turn to face that direction and stay facing that direction.
- `TRACKING#` - The camera will start panning back and forth between minimum and maximum angles

- `GET_FRAME#` - The controller will respond to the message by transmitting 48\*36 bytes over TCP, consisting of the bytes of the latest raw frame from the camera

## Normal operation

---

The component will perform the following loop:

1. Lock the mutex
2. Read the latest values (current, min and max angle, the status and the video data) from shared memory
3. Unlock the mutex
4. If motion has been detected (see the 'Motion detection' section for more detail):
  - A. Connect to the overseer
  - B. Send the following message: `CAMERA {id} MOTION#`
  - C. Close the socket
5. If there are any TCP connections waiting:
  - A. Accept the TCP connection and read the next #-terminated message from the overseer
  - B. If the message received is `ON#` and the camera status is currently `O`:
    - I. Close the connection
    - II. Lock the mutex
    - III. Set the camera status to `O`
    - IV. Unlock the mutex
  - C. If the message received is `OFF#` and the camera status is not `-`:
    - I. Close the connection
    - II. Lock the mutex
    - III. Set the camera status to `-`
    - IV. Unlock the mutex
  - D. If the message received is `TURN {angle}#` and the camera status is not `-`:
    - I. Close the connection
    - II. Lock the mutex
    - III. If the camera angle is already `{angle}`, set the camera status to `O`
    - IV. Otherwise, set the camera status to `L` or `R` as necessary to achieve progress towards `{angle}` (see the 'Camera operation' section for more detail) and internally register that the camera is turning towards `{angle}`. Note that receiving subsequent `OFF`, `TURN` or `TRACKING` messages will take the camera out of this mode
    - V. Unlock the mutex
  - E. If the message received is `TRACKING#` and the camera status is not `-`:
    - I. Close the connection
    - II. Lock the mutex
    - III. If the camera status is `O`, let it to `L`. Otherwise, leave it unchanged
    - IV. Unlock the mutex
    - V. Internally register that the camera is in tracking mode. Note that receiving subsequent `OFF` or `TURN` messages will take the camera out of tracking mode
  - F. If the message received is `GET_FRAME#` and the camera status is not `-`:
    - I. Respond with 48\*36 bytes, consisting of each pixel of the video frame in row major (that is, left-to-right, top-to-bottom) order. This can be achieved in a single large `send()`, passing it a pointer to the `uint8_t video[36][48]` and sending the appropriate number of bytes
    - II. Close the connection
  - G. Loop back to 5
6. If the camera is in tracking mode (as a result of a `TRACKING#` message):
  - A. If the current angle is `min_angle` and the status is `L`, lock the mutex and set the status to `R`
  - B. If the current angle is `max_angle` and the status is `R`, lock the mutex and set the status to `L`
7. If the camera is turning towards a particular angle (as a result of a `TURN {angle}#` command):
  - A. If the current angle is `>= {angle}` and the status is `R`, lock the mutex and set the status to `O`
  - B. If the current angle is `<= {angle}` and the status is `L`, lock the mutex and set the status to `O`

(Note- be careful here. Don't just do a direct comparison - refer to 'Camera operation' below because an angle may be greater than or equal to a number but still turning towards it)
8. Lock the mutex (unless it is already locked as a result of the actions taken in 6 and 7).
9. Wait on 'cond'
10. Loop back to 2

## Camera operation

---

Camera angles are represented as integer angles, in degrees, in the range of 0 to 359. The `min_angle` and `max_angle` values are fixed (by the physical placement and characteristics of the camera) and correspond to the **leftmost** and **rightmost** angles the camera can turn to. This means `min_angle` can be higher than `max_angle`. They can also be the same value (if the camera cannot turn at all).

As an example, if `min_angle` is 330 and `max_angle` is 29, that means the camera has a 60 degree cone of vision. In this example, if angle is 330 and the camera status is R, that angle will increase over time, up to 359, then it will wrap around to 0 and continue increases to 29, which is the angle it will stay at. Note that, even upon reaching 29, the camera status will stay R until the camera controller changes it - but the camera angle will not change further as the maximum angle has been reached.

The camera will change its angle automatically over time according to its status - if it is L, the angle will decrease, and if it is R, the angle will increase. The camera will not turn if its status is O or -. The camera will capture frames of infrared video at fixed intervals and store them in the 'video' array of shared memory, where a pixel value of 0 corresponds to least intense / coldest and 255 corresponds to most intense / hottest. If the camera's status is - the camera will store all 0s in that array instead. Either way, the camera will signal the condition variable each time it updates, and updates will be consistent and happen whether the camera is off, on or turning.

## Motion detection

---

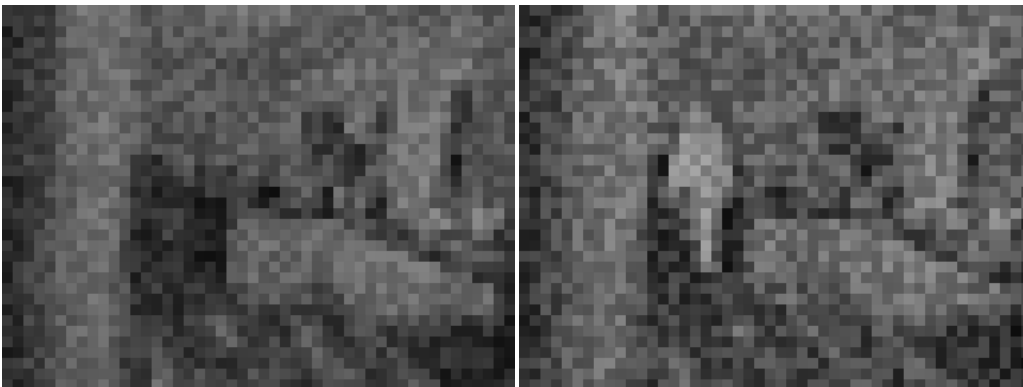
Motion detection is carried out by comparing two frames- the current frame of video captured from the camera and the previous frame captured **at the same angle**. This is because turning cameras might incorrectly detect motion as a result of the turning. This, in practice, means that a camera in tracking mode will not detect any motion until it has made the first sweep, because it has not seen some of those angles before.

Also note that motion detection should not be performed if the previous frame was captured when the camera was off, as there will be a sudden increase in pixel intensity upon the camera turning on, and this does not correspond to motion either.

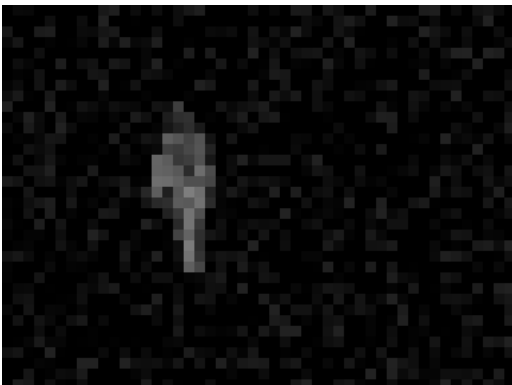
There are multiple steps to detecting motion, once the two frames to compare have been obtained.

1. Subtract the previous frame from the current frame to create a **delta frame** (pixels below 0 become 0 in the delta frame).
2. Compute the pixel values of a new **detection frame** by averaging the values of the 5x5 surrounding pixels (ignoring all pixels that are <2 away from the edge, so the resulting detection frame is a 44x32 image)
3. If the detection frame contains any pixels with values greater than or equal to the temperature change threshold, motion has been detected

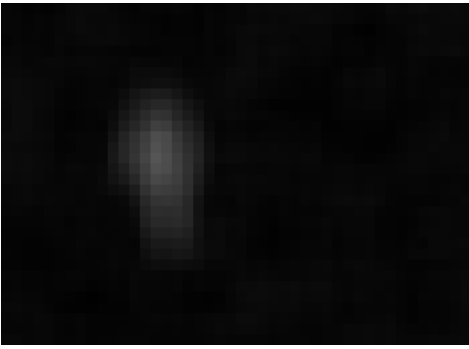
For example, the 'before' and 'after' frames might be these:



By subtracting the former from the latter, the delta frame is created:



Then, to reduce the impact of noise, the 44x32 detection frame is computed:



Whether there is any motion detected depends on whether any of those pixels are greater than or equal to the temperature change threshold.

## Example operation

The program might be executed from the command-line with the following:


```
./camera 444 127.0.0.1:8001 64 /shm 940 127.0.0.1:3000
```

The program will `shm_open` shared memory segment at `/shm` with an offset of 940 and the size of the struct defined above, and `mmap` it into memory. It will also bind TCP port 8001 and begin listening on it. (It puts the socket into nonblocking mode so that it can `accept()` when there are no connections waiting.)

The program will start by connecting to 127.0.0.1:3000 via TCP and sending the following message: `CAMERA 444 127.0.0.1:8001 HELLO#` before disconnecting.

It will then lock the mutex, retrieve the camera status from shared memory, then unlock the mutex.


For this example, let's say the values in shared memory are as follows:

```
{
    uint16_t current_angle = 45;
    uint16_t min_angle = 0;
    uint16_t max_angle = 90;
    char status = '0';
    uint8_t video[36][48] = ;
};
```

This means the camera is currently on, with an angle of 45 degrees, and stationary (not turning left or right). As this is the camera's first time seeing a frame captured at 45 degrees, it cannot detect any motion (as it has no basis for comparison)- however, it does store that frame for later.

Assuming there are no connections waiting (`accept()` returns -1 and `errno` is `EAGAIN` or `EWouldBlock`), the controller will lock the mutex again and then wait on the condition variable 'cond' for the next frame to be generated.

After waking up, the controller then reads the struct again:

```
{
    uint16_t current_angle = 45;
    uint16_t min_angle = 0;
    uint16_t max_angle = 90;
    char status = '0';
    uint8_t video[36][48] = ;
};
```

Nothing has changed other than the video pixels, but as the controller has already seen a frame at 45 degrees it can check for motion, so it runs the two frames through the motion detection algorithm (described above). In this case, the resulting detection frame has a number of pixels with an intensity  $\geq 64$ , so the program connects to 127.0.0.1:3000 via TCP and sends the following message: `CAMERA 444 MOTION#`, then disconnects.

It then continues by checking for incoming TCP connections again.