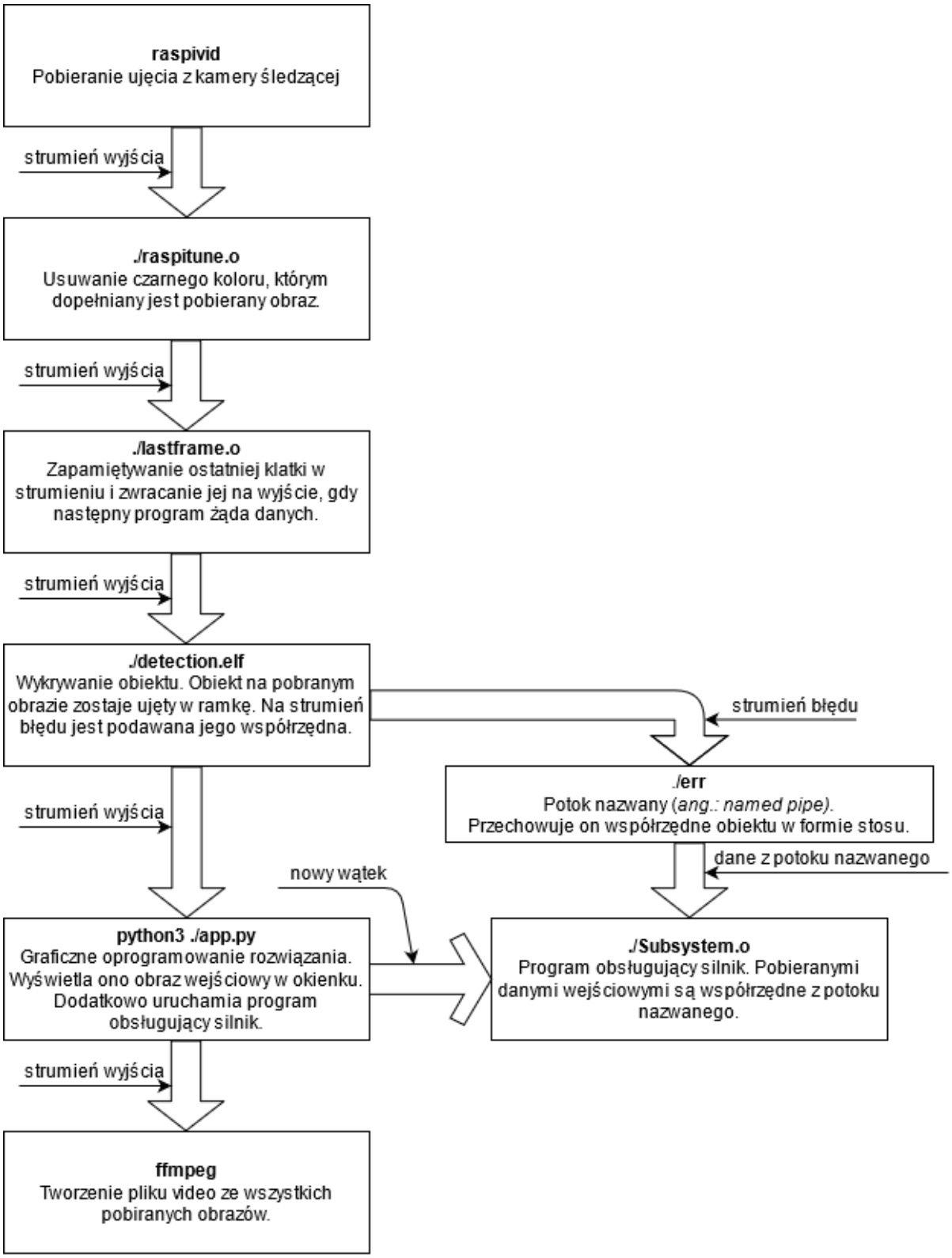
6. Wdrożenie na Raspberry Pi

Całościowe rozwiązanie jest skryptem w powłoce *BASH*. Jego trzon stanowi polecenie potokujące strumień danych, którego źródłem jest wideo z kamerki. Każdy kolejny program w potoku otrzymuje klatkę od poprzedniego i wysyła je następnemu w tej samej lub zmienionej formie. Ostatni z nich ma za zadanie stworzyć filmik ze wszystkich ujęć, które uzyskał na swoim wejściu. Takie podejście pozwala na wyraźne oddzielenie każdego etapu w całym procesie działania, łatwą możliwość edycji każdego z nich bez wpływu na pozostałe, a także możliwość użycia dowolnego sposobu jego wdrażania. Rozwiązanie będzie przyjmowało dwa argumenty stanowiące szerokość i wysokość obrazu wyrażoną w pikselach. Z tych dwóch wartości korzysta każdy program w potoku powodując ich elastyczność pod względem rozdzielczości przetwarzanego obrazu.

```
#!/bin/bash

#EXAMPLE USAGE: ./main.sh 640 480
WIDTH=$1 #Width of image in pixels
HEIGHT=$2 #Height of image in pixels
FPS=40 #Frames per second
NOW=`date +'D%Y-%m-%dT%H-%M-%S'` #Current datetime as string

cd /home/pi/projektpythongui #Make callable from any place of disk

raspivid -n -hf -rot 90 -awb horizon -fps $FPS -w $WIDTH -h $HEIGHT -t 0
--raw - -rf rgb -o /dev/null  |
    #./raspitune.o $WIDTH $HEIGHT |
    ./lastframe.o $WIDTH $HEIGHT 3 |
    ./detection.elf 2> err|
    python3 ./app.py $WIDTH $HEIGHT |
    ffmpeg  -r $FPS -s ${WIDTH}x${HEIGHT} -f ppm_pipe -i -
video${NOW}.avi
```

Kod źródłowy 3. Skrypt programu w powłoce BASH.

Źródłem danych wejściowych jest gotowy program *raspivid*. Jego opcje zostały tak skonfigurowane, aby przekazywany przez niego obraz był dostarczany w zadanej rozdzielczości i z jak największą częstotliwością. Wybranym formatem obrazu jest RGB24. Gwarantuje on, że każda klatka wysyłana przez strumień zawiera identyczną ilość danych. Dzięki temu w prosty sposób można rozróżnić kiedy kończy się jedno ujęcie i zaczyna kolejne. Ponadto umożliwia on obrócenie obrazu o dowolny kąt, który został tak dobrany, aby każda klatka wideo pobierana z kamerki zamontowanej w z góry ustalony sposób wyświetlała się prawidłowo [30].



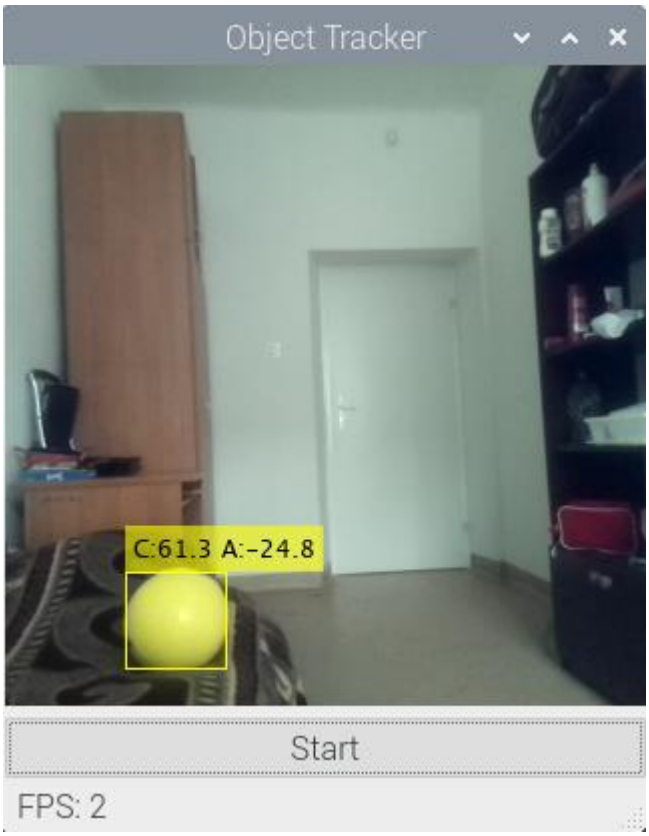
Rysunek 16: Schemat działania rozwiązania.

Następny, nazwany *raspitune* napisany w języku C, rozwiązuje niepożądaną właściwość poprzedniego. Mianowicie *raspivid* wymaga, aby szerokość obrazu wyrażona

w pikselach była podzielna przez 32, a jego wysokość przez 16. Gdy ten warunek nie jest spełniony to ujęcie jest wówczas wypełniane czarnym kolorem po prawej i dolnej stronie. To powoduje, że obraz może mieć oprócz innego wyglądu także inną, większą rozdzielczość. Jego działanie polega na pobieraniu każdego pełnego wiersza obrazu uzupełnionego czarnym kolorem i wysyłaniu na wyjście wersji skróconej o te piksele. Ponadto dodatkowe wiersze na dole ujęcia są przez niego pobierane, ale już nie wysyłane. Wynikowo gwarantuje on, że dostarczany dalej obraz zawsze posiada wymaganą rozdzielczość. Warto zaznaczyć, że gdy obostrzenia *raspivid* są spełnione to ten program nic nie wnosi do działania całości, a on sam w potokowaniu może zostać pominięty.

Obraz zwracany na tym etapie jest już gotowy do wykrywania na nim obiektu. Należy jednak wziąć pod uwagę, że czas detekcji obiektu na jednym ujęciu może być dłuższy od czasu w jakim jest on dostarczany. Jeśli rozwiązanie ma działać w czasie rzeczywistym to należy zadbać o to, aby pobierana klatka przez program wykrywający zawsze była najbardziej aktualną, a więc ostatnią w całości uchwyconą przez matrycę. Ponadto w sytuacji odwrotnej to samo ujęcie nie powinno zostać wykorzystane wielokrotnie. Tę kwestię rozwiązuje kolejny program obsługujący potok danych nazwany *lastframe* napisany w języku C. Korzysta on z dwóch wątków. Pierwszy z nich na bieżąco pobiera ujęcia i zapamiętuje tylko ostatnie z nich. Drugi zaś czeka na żądanie danych na wyjściu za pomocą funkcji *poll* zawartej w pliku nagłówkowym <*poll.h*> [31]. Gdy ono nastąpi zwraca ostatnio zapamiętany obraz. Ponadto program używa dwie blokady obsługujące sekcje krytyczne za pomocą mechanizmu wzajemnego wykluczania (ang.: *mutual exclusion*) [32]. Pierwsza z nich jest uruchamiana gdy wysyłanie danych się rozpocznie lub gdy zaczęło się pobieranie nowej klatki do pamięci. Odblokowanie następuje po ukończeniu któregoś z tych procesów. Zapobiega on zmienianiu danych podczas ich dostarczania na wyjście. Druga z nich jest uruchamiana po wysłaniu obrazu i jest zwalniana po pobraniu kolejnej klatki. Dzięki temu mechanizmowi po wysłaniu jednego ujęcia na jego wyjście program będzie czekał na pobranie kolejnego co uniemożliwi wielokrotne wysyłanie tego samego obrazu.

Kolejnym jest program do rozpoznawania obiektów opisany podrozdziale (Tworzenie programu wykrywającego.) stanowiący istotę całego rozwiązania. Przetworzony przez niego obraz wejściowy z zaznaczonym celem oraz adnotacją jest wysyłany na wyjście. Strumień błędu zawierający współrzędne obiektu przekierowywany jest do potoku nazwanego (*named pipe*) [33]. Posłuży on jako stos danych dla algorytmu obsługującego silnik krokowy, którego sposób uruchomienia zostanie objaśniony w poniższym akapicie.



Rysunek 17. Wykryty obiekt z adnotacją o pewności wykrytego obiektu (C) wyrażonej w procentach oraz współrzędną azymutalną (A). Na samym dole pod przyciskiem zamieszczona informacja o częstotliwości próbkowania (FPS). Przycisk „Start" rozpoczyna śledzenie wykrytego obiektu.

Następnym programem jest aplikacja napisana w języku *Python* stanowiąca przede wszystkim interfejs graficzny dla rozwiązania. W okienku jest przedstawiony podgląd obrazów dostarczanych na jego wejście. Dodatkowo w pasku na dole jest przedstawiona informacja o ilości przetwarzanych klatek na sekundę, a na środku przycisk uruchamiający program obracający mechanizmem. Jego uruchomienie odbywa się poprzez rozpoczęcie podprocesu wewnątrz opisywanej tutaj aplikacji, na którego wejście są przekierowane dane z potoku nazwanego, który został wspomniany w akapicie wyżej. Działanie całego rozwiązania śledzącego obiekt może zostać zakończone poprzez zamknięcie okna aplikacji. Zanim to nastąpi jest jeszcze wysyłany sygnał do uruchomionego tutaj programu sterującego silnikiem, który ma za zadanie umożliwić mu zakończenie jego działania w ustalony przez niego sposób. Na wyjście są wysyłane obrazy wejściowe w formacie PPM w wersji binarnej [34] na potrzeby kolejnego programu w łańcuchu potokowania.

Ostatnim użytym programem jest *ffmpeg* będący gotowym rozwiązaniem, który realizuje tworzenie wideo z obrazów. W jego opcjach została określona rozdzielczość, ilość klatek na sekundę, format danych wejściowych- strumień obrazów w formacie PPM oraz nazwa pliku, do którego wideo ma być zapisane [35]. Jej rozszerzenie decyduje o formacie wideo, który ma być zastosowany, a ona sama stanowi aktualną datę i czas.