Università di Pisa

Corso di Laurea in Ingegneria Robotica
e dell'Automazione

Progetto di Sistemi di Guida e Navigazione

Utilizzo di un drone Crazyflie per l'inseguimento del moto di una Active-Wand avvalendosi del sistema di visione Vicon

Studenti:

Alessio Fornaciari, Giacomo Fanfani

Indice

Presentazione Hardware

Introduzione

Questa relazione è stata da noi pensata non solo come strumento di sintesi di quanto da noi fatto in merito a questo progetto ma anche come possibile "manuale" per chi dopo di noi vorrà continuare il lavoro svolto fino ad ora. Pertanto per iniziare forniremo una breve descrizione dell'hardware da noi utilizzato durante tutto il progetto.

Drone Crazyflie

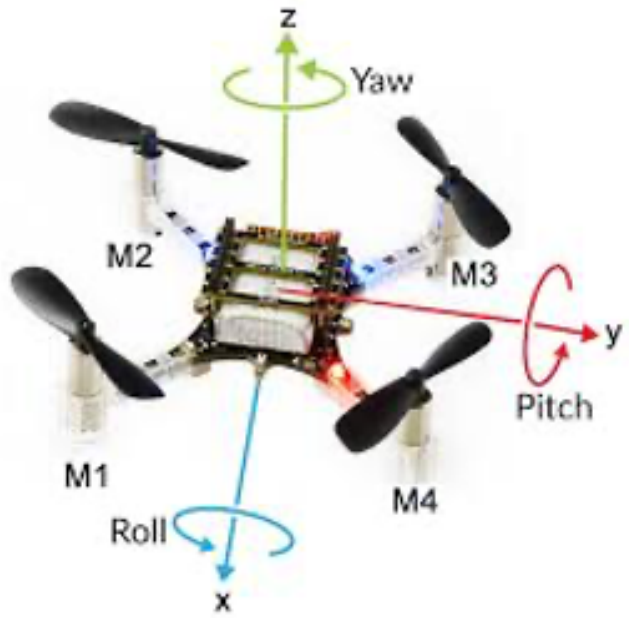


Figura 1: Drone e relativo sistema di riferimento Body

Il nostro drone è un Crazyflie prodotto da BitCraze del tutto equivalente a quello mostrato sopra con l'aggiunta del fatto che sul nostro abbiamo montato anche una piattaforma nella

parte superiore che permettesse il posizionamento dei marker, indispensabili per permettere l'individuazione del Drone da parte delle telecamere, e che mostreremo di seguito.

Markers

Figura 2: Piattaforma per posizionamento Markers

Al fine di permettere la corretta individuazione del Drone da parte delle telecamere, abbiamo sviluppato un'apposita piattaforma che è stata poi stampata e apposta sulla parte superiore al fine di permettere un migliore posizionamento dei marker. Dato che marker troppo vicini davano fastidio al sistema di visione, abbiamo scelto una piattaforma che si estendesse maggiormente in ampiezza rispetto a quella originale grazie ai prolungamenti che riescono a passare tra le eliche senza dare loro fastidio durante il volo. Il colore nero è preferibile in quanto tra tutti è quello che meno riflette e che causa quindi minori interferenze con le telecamere. La disposizione dei maker è arbitraria purchè siano fissati in modo rigido e non crei una geometria simmetrica, ciò è necessario in quanto durante il movimento del Drone il sistema di visione deve poter riconoscere l'orientamento del Drone senza ambiguità, non si devono dunque verificare casi in cui

la disposizione dei marker potrebbe corrispondere a due o più diverse configurazioni del Drone, in quel caso infatti il sistema non riuscirebbe a riconoscere l'orientazione corretta del Drone e fornirebbe quindi valori di orientazione errati rispetto a quelli reali.

Eliche

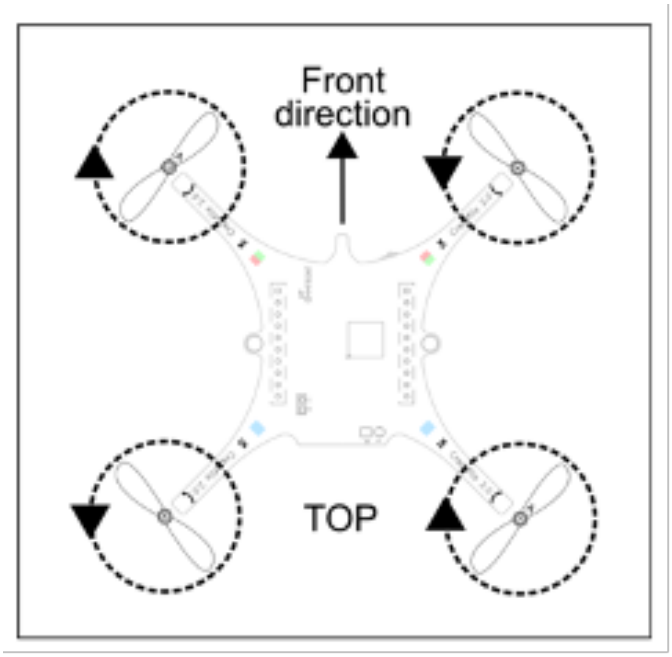


Figura 3: Corretto posizionamento delle eliche

Dato che può capitare che un esperimento non vada come previsto e termini con una caduta del Drone, una breve parentesi sul corretto montaggio delle eliche è necessaria in quanto potrebbe succedere che nella caduta alcune si stacchino dal relativo albero. Nel caso in qui questo succeda è necessario tenere a mente che le eliche sono di due tipi e che, come indica la figura, il montaggio non è del tutto arbitrario ma deve seguire la seguente logica:

- `Eliche di tipo A2:` queste devono essere montate sui motori il cui verso di rotazione è quello orario.
- `Eliche di tipo B2:` al contrario, devono essere montate sui motori il cui verso di rotazione è quello antiorario.

Si fa notare che le eliche dello stesso tipo saranno alla fine disposte l'una di fronte all'altra (non accanto) e che il verso di rotazione di ciascun motore è riportato sui terminali del drone come ultimo simbolo.
Nel momento in cui queste vengono rimontate è bene fare una pressione abbastanza forte in modo da assicurarsi che non si stacchino più con facilità.

Crazyradio

Figura 4: Crazyradio

Per poter comunicare il Drone utilizza il protocollo CRTP appositamente sviluppato da Bitcraze (verrà descritto nel dettaglio più avanti).
La Crazyradio è il mezzo che consente la comunicazione bidirezionale tra il pc su cui essa è installata e il Drone.

Vicon Active Wand

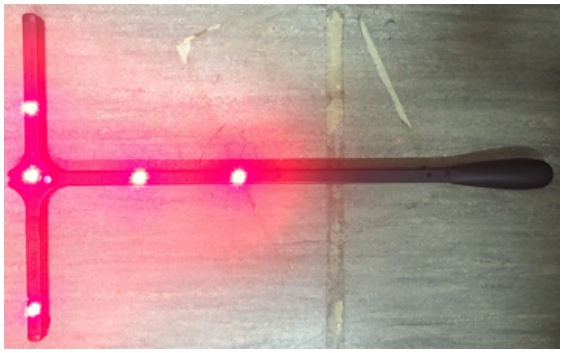


Figura 5: Vicon Active Wand in funzione

Questa è la "Bacchetta" da noi utilizzata come target da inseguire. Come si nota ha già installati dei piccoli laser che una volta accesi fungono da marker. A differenza dei marker del drone (passivi) questi sono marker attivi e dunque conenstono un tracciamento molto più preciso da parte delle telecamere.
A volte è anche utilizzata per ricalibrare l'intero sistema e fornire posizione e orientazione della

terna fissa (da noi rinominata come "terna Vicon"). La ricalibrazione è consigliata all'inizio di ogni sessione di esperimenti, in alternativa, ogni volta che si verificano cambiamenti in termini di "fantasie" di pavimentazione, luce, temperatura ... Diventa praticamente obbligatoria invece nel momento in cui il Software non riesce a visualizzare correttamente i marker (attivi o passivi) posti sugli oggetti, può anche accadere che ne vengano mostrati più di quanti in realtà sono presenti sui relativi oggetti.

Vicon Tracker

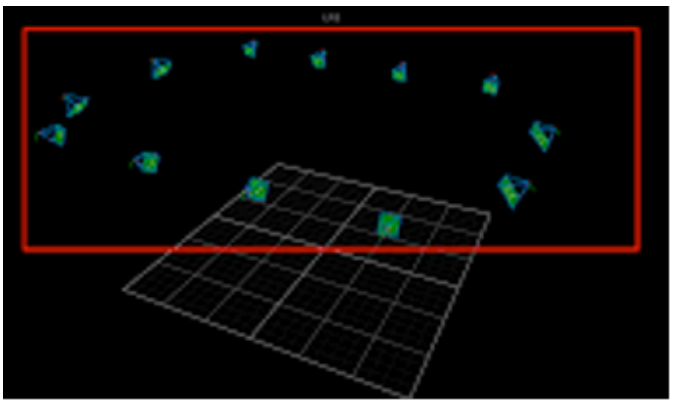


Figura 6: Software Tracker in funzione con evidenziate in rosso le telecamere disposte nella flight room, il cui pavimento è rappresentato dalla griglia.

Il Sistema di tracciamento Vicon è in realtà costituito sia da Hardware che da Software, più precisamente il Software di Tracking si avvale delle telecamere, posizionate nella flight room, per individuare i Markers posti sugli oggetti.
In figura si mostrano le telecamere così come appaiono dal software.
Inizialmente (e periodicamente) il sistema ha bisogno di una ricalibrazione grazie alla quale, con l'ausilio della Wand, viene fatto un reset delle impostazioni delle telecamere per renderle pù precise; vengono inoltre reimpostate sia l'origine della terna fissa (solitamente posta al centro della stanza) che l'orientazione dei relativi assi.
Una volta calibrato, il sistema deve essere in grado di individuare in modo esatto i markers piazzati sugli oggetti. Questi possono essere selezionati in gruppo per creare degli "oggetti software" da associare ai "dispositivi fisici". Tra le proprietà (campi) più utili di questi oggetti troviamo il nome, utilizzato come "alias" all'interno del codice Python, e i sistemi di riferimento "body", che devono essere opportunamente inzializzati con posizione e orientazione desiderati (in alternativa il sistema li inizializza nel modo che sul momento ritiene più adatto, è dunque sempre consigliato di reimpostarli manualmente).

Panoramica generale del Progetto

Scopo del progetto è quello di integrare tutti i dispositivi precedentemente descritti al fine di realizzare l'inseguimento, da parte del Drone, del moto della Wand all'interno della flight-room avvalendosi del sistema di posizionamento Vicon che fornisce sia la misura di "posizione target" della Wand che la misura di posizione del Drone, quest'ultima da utilizzare in ingresso al filtro di Kalman del Drone stesso per l'aggiornamento della sua posizione corrente.
In realtà il sistema Vicon fornisce anche misure di orientazione (fornite tramite relativi quaternioni) che però per quanto riguarda la Wand non sono di interesse per gli scopi di questo progetto mentre, per quanto riguarda il drone, sono stati oggetto di molte sperimentazioni che purtroppo ad oggi non hanno ancora consentito di utilizzarli in modo "completo", in particolare il progetto allo stato attuale utilizza le misure di orientazione del Drone soltanto per effettuare delle conversioni tra i sisitemi di riferimento utilizzati, non prende invece in considerazione tali misure come ingressi al filtro di Kalman del Drone.

Di seguito presentiamo uno schema concettuale con le connessioni stabilite tra i vari componenti, accompagnato da una breve descrizione:

- Le telecamere, tramite l'individuazione dei marker posizionati sugli oggetti, permettono al Tracker di ricavare in ogni istante posizioni e orientazioni di Drone e Wand;
- Tramite le funzioni della libreria Vicon-DSSDK il programma in esecuzione riceve dal tracker le posizioni e le orientazioni di interesse;
- Il programma elabora posizioni e orientazioni effettuando eventuali conversioni tra sistemi di riferimento e tramite la Crazyradio e le librerie Python che implementano il protocollo CRTP:
  - Fornisce al Drone i nuovi riferimenti di posizione da inseguire, ricavati dalla posizione aggiornata della Wand;
  - Fornisce al Drone le nuove misure della sua posizione da utilizzare per l'aggiornamento del filtro di Kalman.
- E cosi via ..

(più avanti dedicheremo un paragrafo per scendere nel dettaglio di tutte le operazioni necessarie)

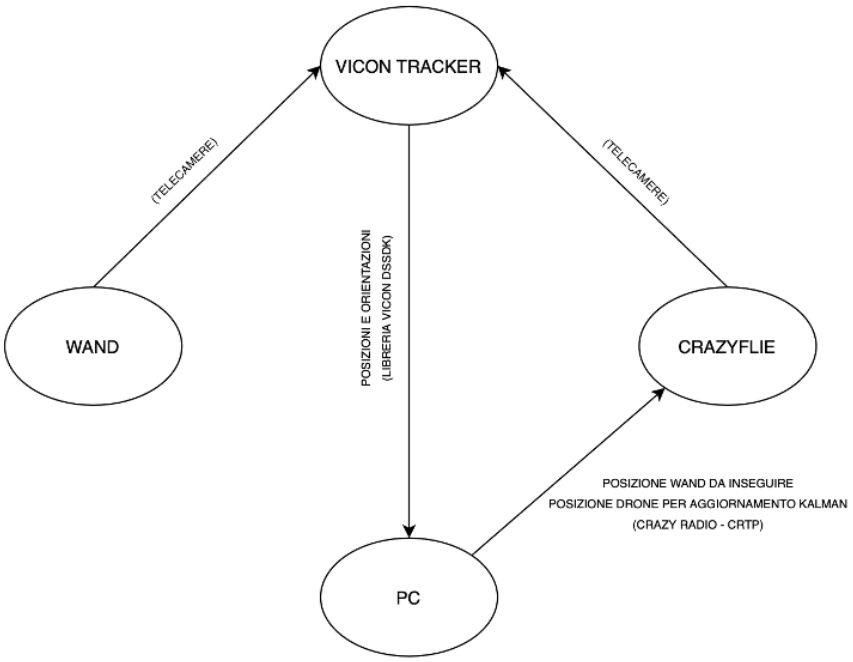


Figura 7: Schema a blocchi delle entità coinvolte nel progetto

Breve Panoramica sul Protocollo CRTP

Il protocollo CRTP (Crazy Real Time Protocol) è il protocollo creato appositamente da Bitcraze per gestire la comunicazione da/verso il Drone.
Concettualmente la comunicazione che avviene tra il Drone e la Crazyradio segue il seguente schema:

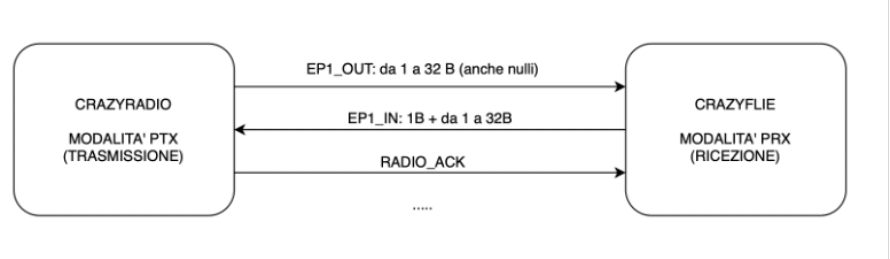


Figura 8: Handshake CRTP

Le relative implementazioni Python dei due blocchi sopra mostrati sono le seguenti:

ptx.py

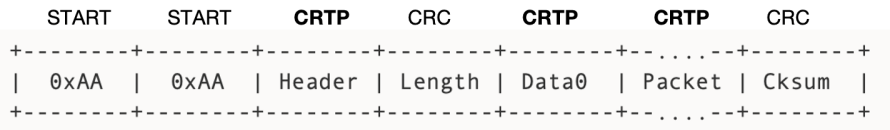
```
import sys
sys.path.append("../lib")
from crazyradio import Crazyradio

radio = Crazyradio()
radio.set_channel(100)
radio.set_data_rate(Crazyradio.DR_250KPS)
radio.set_mode(Crazyradio.MODE_PTX)
for i in range(0, 100):
    res = radio.send_packet([i])
    print(res.data)
radio.close()
```

prx.py

```
import sys
sys.path.append("../lib")
from crazyradio import Crazyradio

radio = Crazyradio()
radio.set_channel(100)
radio.set_data_rate(Crazyradio.DR_250KPS)
radio.set_mode(Crazyradio.MODE_PRX)
for i in range(0, 100):
    res = radio.receive()
    if res:
        radio.sendAck([i])
    print(res)
radio.close()
```

Figura 9: Handshake CRTP con funzioni Python

Andando più nel dettaglio abbiamo visto come questo protocollo si avvale di pacchetti che hanno la seguente struttura:



Field	Byte	Bit	Description
Start	0-1	0-2	Start token for synchronization (0xAAAA)
Header	2	0-1	The destination channel
	2	2-3	Reserved for the link layer
	2	4-7	The destination port
Size	3		Number of data byte
Data	4-..		The data in the packet
Cksum	4+size		Checksum. Sum of all the bytes, excluding *Start*, modulo 256.

Figura 10: Pacchetti CRTP - Specifichiamo che il campo Data può contenere al massimo 31B

Un'ulteriore analisi della codifica della Porta di destinazione è necessaria:

Port	Target	Used for
0	Console	Read console text that is printed to the console on the Crazyflie using consoleprintf
2	Parameters	Get/set parameters from the Crazyflie. Parameters are defined using a macro in the Crazyflie source-code
3	Commander	Sending control set-points for the roll/pitch/yaw/thrust regulators
4	Memory access	Accessing non-volatile memories like 1-wire and I2C (only supported for Crazyflie 2.0)
5	Data logging	Set up log blocks with variables that will be sent back to the Crazyflie at a specified period. Log variables are defined using a macro in the Crazyflie source-code
6	Localization	Packets related to localization
7	Generic Setpoint	Allows to send setpoint and control modes
13	Platform	Used for misc platform control, like debugging and power off
14	Client-side debugging	Debugging the UI and exists only in the Crazyflie Python API and not in the Crazyflie itself.
15	Link layer	Used to control and query the communication link

Figura 11: Codifica porte di destinazione dei pacchetti CRTP

Di seguito riportiamo un banale esempio in cui vediamo come dovrebbe essere fatto un pacchetto per voler inviare al `Commander` (porta 3) un campo `Data` contenente riferimenti nulli di orientazione e spinta dei motori:

To send setpoints to the commander send:

```
0xaa 0xaa 0x30 0x0e 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x3e
```

```
        +-------+-------+-------+-------+
        | ROLL  | PITCH |  YAW  |THRUST |
        +-------+-------+-------+-------+
Length      4       4       4       2      bytes
```

Name	Byte	Size	Type	Comment
ROLL	0-3	4	float	The pitch set-point
PITCH	4-7	4	float	The roll set-point
YAW	8-11	4	float	The yaw set-point
THRUST	12-13	2	uint16_t	The thrust set-point

Figura 12: Esempio di invio pacchetto CRTP

Tornando allo scopo del progetto, è per noi di interesse la porta 6 utilizzata per la trasmissione di pacchetti contenenti informazioni di posizione e orientazione.
Di seguito vediamo come, tramite la codifica di due diversi "canali", possiamo utilizzare questa porta per la trasmissione di informazioni automaticamente indirizzate al filtro di Kalman (utilzzando il **canale 0**):

This port groups various packets related to localization. It exposes two channels:

Port	Channel	Name
6	0	External Position
6	1	Generic localization

External Position

This packet is used to send the Crazyflie position as acquired by an external system.
acquired by a motion capture system to push it in the Extended Kalman Filter to allc
estimate and control its state.

The packet format is:

```
struct CrtpExtPosition
{
  float x; // in m
  float y; // in m
  float z; // in m
} __attribute__((packed));
```

Figura 13: External Position

Qualora volessimo trasmettere anche l'informazione dell'orientazione, in aggiunta a quella di posizione, dobbiamo avvalerci del **canale 1** che consente l'invio di una struttura dati di dimensioni maggiori, come mostrato di seguito:

Generic Localization

This channel intends to host packets useful for the localization subsystem. It has been created to serve the Loco Positioning System packets but can be used for more general things like GPS NMEA or binary streams. The format of the packet is:

Byte	Value	Note
0	ID	ID of the packet
1..	Payload	Packet payload. Format defined per ID

ID	Packet
2	LPP Short packet tunnel
3	Enable emergency stop
4	Reset emergency stop timeout

```
struct CrtpExtPose
{
  float x; // in m
  float y; // in m
  float z; // in m
  float qx;
  float qy;
  float qz;
  float qw;
} __attribute__((packed));
```

Figura 14: External Position

Breve panoramica sulla libreria CFLIB - Python

Riferimenti di Posizione

Questa libreria è stata appositamente sviluppata da Bitcraze e contiene tutte le funzioni necessarie per comunicare da/verso il Drone. Essendo open-source al suo sviluppo può contribuire chiunque e c'è molta disponibilità di materiale sul web.
La libreria contiene svariate classi grazie a cui si possono inviare comandi al Drone, di seguito a titolo di esempio e soltanto per fornire un'idea di come queste funzioni sono implementate, mostreremo alcune delle funzioni e andremo ad analizzarle passando dal vedere come queste sono disponibili "ad alto livello" e guardando come si implementano "a basso livello" fino alle funzioni che implementano l'invio dei pacchetti CRTP spiegati in precedenza.
Un esempio è la classe `Commander` che consente di inviare al Drone riferimenti di posizione da inseguire e che è la stessa utilizzata nel nostro codice. Ne vediamo un estratto in figura 15.
In realtà per essere più precisi specifichiamo che nel codice utilizzeremo anche una variante della classe `Commander` e che si chiama `MotionCommander` . La particolarità di quest'ultima è che consente di effettuare la procedura di decollo in modo "automatico" una volta creato il relativo oggetto Dorne. Avremo dunque nel nostro codice un oggetto per ognuna di queste due classi ma utilizzeremo questa variante soltanto per effettuare il decollo mentre per l'intera parte rimanente ci avvaleremo della classe `Commander`.

Al loro interno queste funzioni consentono di costruire i pacchetti CRTP a partire dai valori dei riferimenti desiderati. L'invio del pacchetto è fatto in accordo con quanto previsto dal protocllo CRTP utilizzando la send_packet riportata in figura 16.

A sua volta questa funzione ricorre alle funzioni proprie del driver della Crazyradio. In figura 17 riportiamo la send_packet che di fatto consente alla Crazyradio di inviare il pacchetto CRTP (notiamo come questa, essendo parte del firmware è scritta in C).

```python
class Commander():
    """
    Used for sending control setpoints to the Crazyflie
    """

    def __init__(self, crazyflie=None):
        """
        Initialize the commander object. By default the commander is in
        +-mode (not x-mode).
        """
        self._cf = crazyflie
        self._x_mode = False

    def send_position_setpoint(self, x, y, z, yaw):
        """
        Control mode where the position is sent as absolute x,y,z coordinate in
        meter and the yaw is the absolute orientation.

        x and y are in m
        yaw is in degrees
        """
        pk = CRTPPacket()
        pk.port = CRTPPort.COMMANDER_GENERIC
        pk.data = struct.pack('<Bffff', TYPE_POSITION,
                              x, y, z, yaw)
        self._cf.send_packet(pk)
```

Figura 15: Classe Commander

```
def send_packet(self, pk, expected_reply=(), resend=False, timeout=0.2):
    """
    Send a packet through the link interface.

    pk -- Packet to send
    expect_answer -- True if a packet from the Crazyflie is expected to
                     be sent back, otherwise false

    """
    self._send_lock.acquire()
    if self.link is not None:
        if len(expected_reply) > 0 and not resend and \
                self.link.needs_resending:
            pattern = (pk.header,) + expected_reply
            logger.debug(
                'Sending packet and expecting the %s pattern back',
                pattern)
            new_timer = Timer(timeout,
                              lambda: self._no_answer_do_retry(pk,
                                                               pattern))
            self._answer_patterns[pattern] = new_timer
            new_timer.start()
        elif resend:
            # Check if we have gotten an answer, if not try again
            pattern = expected_reply
            if pattern in self._answer_patterns:
                logger.debug('We want to resend and the pattern is there')
                if self._answer_patterns[pattern]:
                    new_timer = Timer(timeout,
                                      lambda:
                                      self._no_answer_do_retry(
                                          pk, pattern))
                    self._answer_patterns[pattern] = new_timer
                    new_timer.start()
            else:
                logger.debug('Resend requested, but no pattern found: %s',
                             self._answer_patterns)
        self.link.send_packet(pk)
        self.packet_sent.call(pk)
    self._send_lock.release()
```

Figura 16: send_packet - Chiamata all'interno della send_setpoint responsabile del solo handshake iniziale

```
def send_packet(self, dataOut):
    """ Send a packet and receive the ack from the radio dongle
        The ack contains information about the packet transmition
        and a data payload if the ack packet contained any """
    ackIn = None
    data = None
    try:
        if (pyusb1 is False):
            self.handle.bulkWrite(1, dataOut, 1000)
            data = self.handle.bulkRead(0x81, 64, 1000)
        else:
            self.handle.write(endpoint=1, data=dataOut, timeout=1000)
            data = self.handle.read(0x81, 64, timeout=1000)
    except usb.USBError:
        pass

    if data is not None:
        ackIn = _radio_ack()
        if data[0] != 0:
            ackIn.ack = (data[0] & 0x01) != 0
            ackIn.powerDet = (data[0] & 0x02) != 0
            ackIn.retry = data[0] >> 4
            ackIn.data = data[1:]
        else:
            ackIn.retry = self.arc

    return ackIn
```

```
0x81,                   //bEndpointAddess (EP1 IN)
```

Figura 17: send_packet - Tramite la bulkWrite invia dataOut per poi attendere quanto ricevuto dal Drone e metterlo in data

Aggiornamento filtro di Kalman

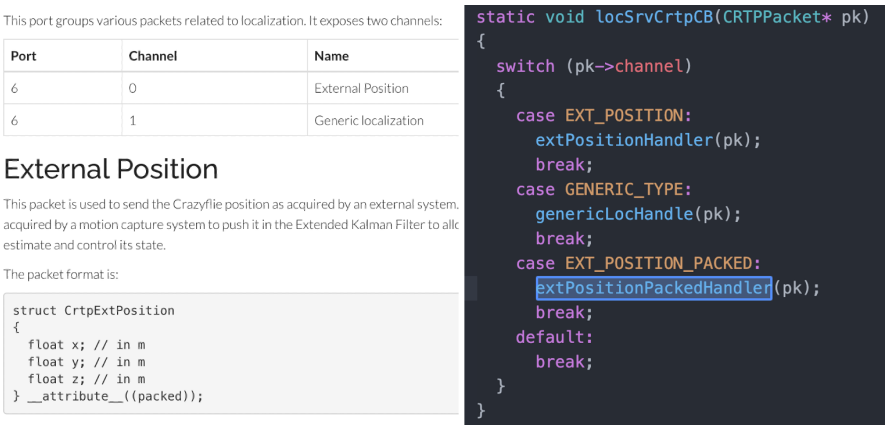
Per inviare le misure di posizione al filtro di Kalman utilizziamo la funzione `send_extpos` che riportiamo in figura 18.

```
class Extpos():
    """
    Used for sending its position to the Crazyflie
    """

    def send_extpos(self, x, y, z):
        """
        Send the current Crazyflie X, Y, Z position. This is going to be
        forwarded to the Crazyflie's position estimator.
        """

        self._cf.loc.send_extpos([x, y, z])

    def send_extpose(self, x, y, z, qx, qy, qz, qw):
        """
        Send the current Crazyflie X, Y, Z position and attitude as a
        normalized quaternion. This is going to be forwarded to the
        Crazyflie's position estimator.
        """
        self._cf.loc.send_extpose([x, y, z], [qx, qy, qz, qw])
```

Figura 18: In figura sono riportate sia la send_extpos, utilizzata per l'invio della sola posizione, che la send_extpose, utilizzata per l'invio di posizione e orientazione

Per quanto riguarda la gestione dei messaggi nel caso di ricezione (da parte del Drone) di informazioni di posizione da utilizzare per l'aggiornamento del filtro di Kalman, ricordando quanto detto precedentemente, si riporta nelle figure che vanno dalla 19 alla 21 il "percorso" (in termini di funzioni della libreria) seguito dal pacchetto una volta ricevuto e "spacchettato" dal Crazyflie.

In entrambi i casi, sia che si passi dalla `genericPositionHandler` sia che si passi dalla `extPositionHandler`, una volta inseriti nella coda i dati vengono elaborati come in figura 22.



This port groups various packets related to localization. It exposes two channels:

Port	Channel	Name
6	0	External Position
6	1	Generic localization

External Position

This packet is used to send the Crazyflie position as acquired by an external system. acquired by a motion capture system to push it in the Extended Kalman Filter to allc estimate and control its state.

The packet format is:

```
struct CrtpExtPosition
{
  float x; // in m
  float y; // in m
  float z; // in m
} __attribute__((packed));
```

```
static void locSrvCrtpCB(CRTPPacket* pk)
{
  switch (pk->channel)
  {
    case EXT_POSITION:
      extPositionHandler(pk);
      break;
    case GENERIC_TYPE:
      genericLocHandle(pk);
      break;
    case EXT_POSITION_PACKED:
      extPositionPackedHandler(pk);
      break;
    default:
      break;
  }
}
```

Figura 19: a seconda del canale scritto nel pacchetto l'informazione viene gestita da funzioni diverse che, a seconda del caso, operano secondo quanto stabilito dal protocollo CRTP

```
static void extPositionPackedHandler(CRTPPacket* pk)
{
  uint8_t numItems = pk->size / sizeof(extPositionPackedItem);
  for (uint8_t i = 0; i < numItems; ++i) {
    const extPositionPackedItem* item = (const extPositionPackedItem*)&pk->data[i * sizeof(extPosi
    ext_pos.x = item->x / 1000.0f;
    ext_pos.y = item->y / 1000.0f;
    ext_pos.z = item->z / 1000.0f;
    ext_pos.stdDev = extPosStdDev;
    if (item->id == my_id) {
      estimatorEnqueuePosition(&ext_pos);
      tickOfLastPacket = xTaskGetTickCount();
    }
    else {
      peerLocalizationTellPosition(item->id, &ext_pos);
    }
  }
}
```

Figura 20: Qui notiamo l'utilizzo dell'informazione della sola posizione per assegnarla al valore attuale di posizione del Drone e la messa in coda tra i dati da utilizzare per il filtro di Kalman.

Figura 21: Qui vediamo invece come viene gestita l'informazione quando riceviamo, oltre alla posizione, anche l'orientazione. Il ramo di codice di interesse è quello relativo a "type == EXT_POSE" (in cui EXT_POSE vale 8).

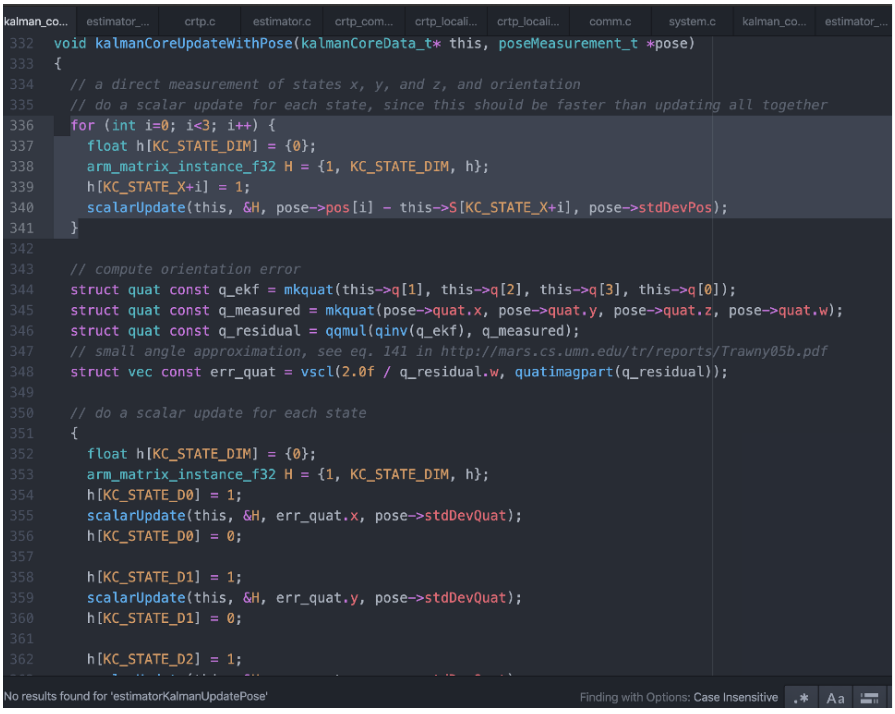

kalman_co...
estimator_...
crtp.c
estimator.c
crtp_com...
crtp_locali...
crtp_locali...
comm.c
system.c
kalman_co...
estimator_...
void kalmanCoreUpdateWithPose(kalmanCoreData_t* this, poseMeasurement_t *pose)
{
// a direct measurement of states x, y, and z, and orientation
// do a scalar update for each state, since this should be faster than updating all together
for (int i=0; i<3; i++) {
float h[KC_STATE_DIM] = {0};
arm_matrix_instance_f32 H = {1, KC_STATE_DIM, h};
h[KC_STATE_X+i] = 1;
scalarUpdate(this, &H, pose->pos[i] - this->S[KC_STATE_X+i], pose->stdDevPos);
}
// compute orientation error
struct quat const q_ekf = mkquat(this->q[1], this->q[2], this->q[3], this->q[0]);
struct quat const q_measured = mkquat(pose->quat.x, pose->quat.y, pose->quat.z, pose->quat.w);
struct quat const q_residual = qqmul(qinv(q_ekf), q_measured);
// small angle approximation, see eq. 141 in http://mars.cs.umn.edu/tr/reports/Trawny05b.pdf
struct vec const err_quat = vscl(2.0f / q_residual.w, quatimagpart(q_residual));
// do a scalar update for each state
{
float h[KC_STATE_DIM] = {0};
arm_matrix_instance_f32 H = {1, KC_STATE_DIM, h};
h[KC_STATE_D0] = 1;
scalarUpdate(this, &H, err_quat.x, pose->stdDevQuat);
h[KC_STATE_D0] = 0;
h[KC_STATE_D1] = 1;
scalarUpdate(this, &H, err_quat.y, pose->stdDevQuat);
h[KC_STATE_D1] = 0;
h[KC_STATE_D2] = 1;
No results found for 'estimatorKalmanUpdatePose'
Finding with Options: Case Insensitive
.*
Aa

Figura 22

Breve panoramica sulla libreria Py-Vicon

Questa libreria è quella che utilizziamo per ottenere dal Tracker le informazioni di posizione e orientazione di Drone e Wand.
Le uniche funzioni per noi di interesse sono quelle riportate nelle figure 23 e 24.
In realtà utilizzeremo anche una terza funzione che ci consente di recuperare l'informazione di orientazione di un oggetto tramite i relativi quaternioni. Questa funzione non è stata riportata in quanto come detto in precedenza il progetto allo stato attuale non "integra" alla perfezione l'utilizzo di questa funzione che sarebbe dovuta servire per aggiornare l'informazione di orientazione del drone nel suo filtro di Kalman.

```
def GetSegmentGlobalRotationEulerXYZ( self, subjectName, segmentName ):
    """Return the rotation of a subject segment in global Euler XYZ coordinates.

        >>> import ViconDataStream
        >>> client = ViconDataStream.Client()
        >>> client.Connect('localhost')
        >>> client.EnableSegmentData()
        >>> client.GetFrame()
        >>> print( client.GetSegmentGlobalRotationEulerXYZ('Alice','Pelvis') )
        ((0.0, 0.0, 0.0), False)

    \throw A DataStreamException class containing the error:
            + NotConnected
            + Success
            + NoFrame
            + InvalidSubjectName
            + InvalidSegmentName

    \return Tuple containing:
            - The rotation of the segment as a 3 touple (X,Y,Z).
            - Occluded will be True if the segment was absent at this frame. In this case the rotation will be [0,0,0].

    See Also : GetSegmentGlobalTranslation, GetSegmentGlobalRotationHelical...
    """
    a = CoreClient.doubleArray( 3 )
    ret, o = self.client.GetSegmentGlobalRotationEulerXYZ( subjectName, segmentName, a )
    if ret != CoreClient.Success:
        raise DataStreamException( ret )
    p = ( a[0], a[1], a[2] )
    return ( p, o )
```

Figura 23: Funzione utilizzata per ricevere dal Tracker la matrice di rotazione che rappresenta l'orientazione di un oggetto rispetto alla terna fissa Vicon

Figura 24: Funzione utilizzata per ricevere dal Tracker il vettore posizione che rappresenta la traslazione dell'origine della terna Body di un oggetto rispetto all'origine della terna fissa Vicon, espressa nel sistema fisso Vicon

Queste funzioni restituiscono rispettivamente una struttura dati da cui poter ottenere facilmente la posizione `[x,y,z]` espressa in terna Vicon e la matrice di rotazione `(3x3)` risultante dalla composizione delle classiche rotazioni successive lungo gli assi `X-Y-Z` (secondo la parametrizzazione `"RPY"`).

Il nostro scopo finale è riuscire ad inviare al Drone:

- Riferimenti di posizione da inseguire utilizzando la `send_position_setpoint(x,y,z,0)` della classe `Commander`. La posizione dovrà essere espressa nella giusta terna. Lo zero finale è il campo relativo all'angolo di Yaw "desiderato" dell'orientazione del Drone. Vedremo più avanti nel dettaglio le motivazioni alla base di questa scelta.
- Informazioni sulla sua posizione attuale all'interno del sistema Vicon al fine di utilizzarle per l'aggiornamento del filtro di Kalman, anche queste inviate dopo averle espresse nella terna corretta. Questo verrà fatto utilizzando la `send_extpos(x,y,z)`.

(Nella sezione successiva riporteremo con più dettaglio tutto quanto riguarda la parte dedicata ai sistemi di riferimento e le convenzioni adottate durante l'esperimento).

Sistemi di Riferimento

Introduzione

Scendendo più nel dettaglio, qui è dove riportiamo le convenzioni che abbiamo adottato sui vari sistemi di riferimento utilizzati durante l'esecuzione degli esperimenti.

Come mostrato in figura 25 abbiamo deciso di adottare le seguenti terne:

- `Sistema Fisso` **Vicon** `(V)` : ha l'origine coincidente col centro della flight room; Gli assi **[** X_v **,** Y_v **,** Z_v**]** sono fissi e rappresentano dunque il frame inerziale.
- `Sistema Fisso` **Start** ($S_\#$) : ha l'origine coincidente con il `CDM` del `Drone` al momento dell'accensione. Ha dunque quota **"z"** nulla e, oltre ad essere complanare al sistema **"V"**, ha anche la sua stessa orientazione. Questo è il frame `navigation`.
- `Sistema Body` **Start** (S_i) : Ha l'origine coincidente con il CDM del Drone in ogni istante t_i. Gli assi sono in ogni momento orientati in modo che l'asse `X` parta dall'origine e vada verso prua, l'asse `Y` parta dall'origine e vada verso sinistra (rispetto all'asse `X)` , l'asse `Z` che va dall'origine verso l'alto (`terna destrorsa).`Rispetto ai due sistemi precedenti, la sua orientazione varia soltanto in termini di Yaw in quanto gli angoli di Pitch e Roll sono sempre supposti pari a zero. In questo modo evitiamo sia di considerare il relativo rumore nella misura di tali angoli da parte del sistema di visione, sia riusciamo a operare con sistemi di riferimento complanari (o "paralleli" nel caso prendano quota `z` > 0).

Per quanto riguarda le misure fornite dal Vicon, queste sono tutte espresse in terna fissa `V`. La parametrizzazione scelta per rappresentare posizione e orientazione del Drone all'interno del sistema `V` è la parametrizzazione **RPY** detta anche **XYZ** .
Dato che il Drone, una volta acceso, crea il suo sistema `Body` in modo che abbia origine e orientazione coincidenti con quelle che assume all'istante di accensione, e dato che considera (per come abbiamo dovuto implementare il progetto) ogni riferimento in ingresso come espresso in un sistema la cui origine coincide con quela del `navigation` ma il cui frame è costantemente ruotato sull'asse `z` di un angolo pari al suo attuale angolo di `yaw` , abbiamo dovuto operare la conversione tra i due sistemi **V** ed S_i (rappresenta il sistema body del drone al generico istante **i**).

Calcolo della matrice di Rotazione da {Vicon} a {Body}

Il Tracker fornisce roll (α) pitch (β) yaw (γ) del drone rispetto al sistema di riferimento fisso Vicon {V}. Costruiamo dunque le varie matrici di rotazione che compongono la parametrizzazione di **Eulero XYZ** :

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} R_y = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} R_z = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In generale, introducendo soltanto per questo paragrafo una notazione più semplice, abbiamo un sistema {V} che costituisce il sistema inerziale e un sistema {D} che costituisce il sistema body del drone. Questo sistema {D} solidale al drone ha il suo asse `X` rivolto da poppa a prua, asse `Y` in modo da avere una terna destrorsa con asse `Z` rivolto verso l'alto. Per riportare un vettore P^v espresso in {V} in un vettore P^d espresso in {D} dobbiamo scrivere la: $P^d = R_{dv} \cdot P^v$ in cui la R_{dv} esprime come passare "graficamente" da {D} a {V} (Per portare in realtà {V} in {D}). Attraverso il prodotto delle 3 matrici otteniamo la matrice R_{xyz} che mappa un vettore in terna {D} in un vettore in terna {V}:

$$R_{xyz} = R_x \cdot R_y \cdot R_z = R_d^v$$

$$R_d^v = \begin{bmatrix} \cos(\gamma)\cos(\beta) & -\cos(\beta)\sin(\gamma) & \sin(\beta) \\ \cos(\alpha)\sin(\gamma) + \cos(\gamma)\sin(\alpha)\sin(\beta) & \cos(\alpha)\cos(\gamma) - \sin(\alpha)\sin(\beta)\sin(\gamma) & -\cos(\beta)\sin(\alpha) \\ \sin(\alpha)\sin(\gamma) - \cos(\alpha)\cos(\gamma)\sin(\beta) & \cos(\gamma)\sin(\alpha) + \cos(\alpha)\sin(\beta)\sin(\gamma) & \cos(\alpha)\cos(\beta) \end{bmatrix}$$

La (R_d^v) per noi esprime come passare "graficamente" da V a D dunque ciò che ci serve è la sua trasposta:

$$R_v^d = R_{zyx} = (R_d^v)^T$$

$$R_v^d = \begin{bmatrix} \cos(\gamma)\cos(\beta) & \cos(\alpha)\sin(\gamma) + \cos(\gamma)\sin(\alpha)\sin(\beta) & \sin(\alpha)\sin(\gamma) - \cos(\alpha)\cos(\gamma)\sin(\beta) \\ -\cos(\beta)\sin(\gamma) & \cos(\alpha)\cos(\gamma) - \sin(\alpha)\sin(\beta)\sin(\gamma) & \cos(\gamma)\sin(\alpha) + \cos(\alpha)\sin(\beta)\sin(\gamma) \\ \sin(\beta) & -\cos(\beta)\sin(\alpha) & \cos(\alpha)\cos(\beta) \end{bmatrix}$$

Panoramica dei Sistemi di Riferimento

La figura 25 rappresenta una panoramica "dall'alto" dunque la componente di traslazione lungo gli assi `z` non si vede ma sappiamo essere presente e necessaria in quanto il `Drone` per traslare dal suo punto di accensione deve aver prima preso quota; pertanto il sistema di riferimento S_1

avrà l'origine sicuramente ad una quota `z` > 0. I sistemi di riferimento $S_{\#}$, S e V sono invece complanari a quota nulla.

In fig. 25 si mostra un esempio in cui il `drone` è inizialmente fermo all'istante t_0 (sistema di riferimento S_0), ha un angolo di `Yaw` pari a - ψ_0 ed è traslato rispetto al centro della stanza di P_{vs} . In realtà ad essere precisi S_0 esprime la reale posa del Drone rispetto all'osservatore nella stanza ma il Drone stesso crede di essere posizionato e orientato come espresso da $S_{\#}$ (quindi con la stessa orientazione della terna V).

Successivamente all'istante t_1 il Drone risulta traslato rispetto al suo punto di accensione di P_{sf} . Il che coincide con il dire che risulta essere traslato rispetto al centro della stanza di P_{vf}. Per tenere presente che negli istanti successivi a quello iniziale il Drone ha una quota **`"z"` > 0** abbiamo indicato nel pedice delle traslazioni la lettera **f** come ultima in quanto rappresentativa della parola **fly**.

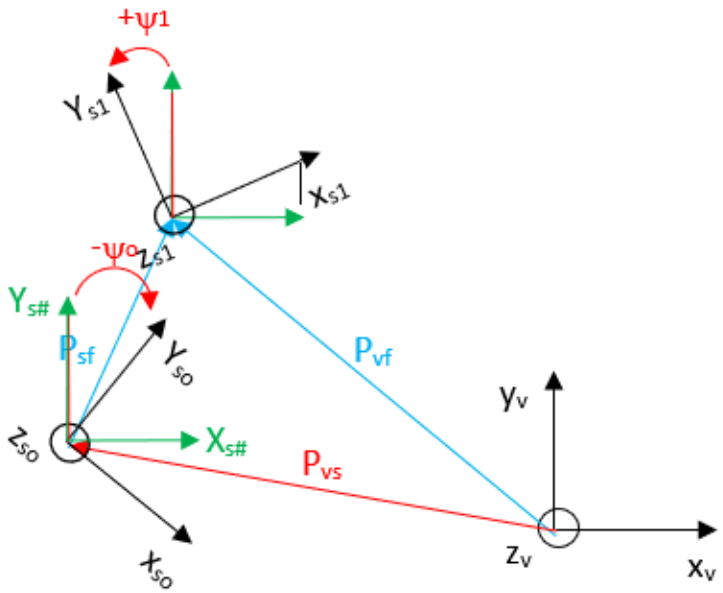


Figura 25: I vari Sistemi di Riferimento introdotti

Andiamo ad elencare i passaggi e le trasformazioni propri del caso rappresentato nella figura 25:

- **Accensione del Drone in** P_{vs} : il `Drone` crea il suo sistema di riferimento Body S_0 orientato come in figura. In realtà il drone in ogni momento dell'esperimento penserà di avere angolo di `yaw` pari a zero in quanto noi non forniamo al filtro di Kalman le nuove misure di orientazione ma soltanto di posizione. Dunque a voler essere precisi, il Drone pensa di essere orientato esattamente come la terna $S_{\#}$.

E' compito nostro (come vedremo di seguito) fornire riferimenti che non siano solo traslati ma anche ruotati dell'angolo di `Yaw` che istante per istante il drone in realtà ha rispetto a V .

- **Creiamo dunque la matrice omogenea di rototraslazione cosi composta**:

  Matrice di Rotazione da $\{S_i\}$ a $\{\mathrm{V}\}$: $R_i^v = R_z(\psi_i) = \begin{bmatrix} \cos(\psi_i) & -\sin(\psi_i) & 0 \\ \sin(\psi_i) & \cos(\psi_i) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

  (ricordiamo che supporremo sempre nulli gli angoli di roll e pitch)

  Matrice di Rotazione da $\{\mathrm{V}\}$ a $\{S_i\}$: $R_v^i = (R_i^v)^T = \begin{bmatrix} \cos(\psi_i) & \sin(\psi_i) & 0 \\ -\sin(\psi_i) & \cos(\psi_i) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

  Traslazione tra origine $\{\mathrm{V}\}$ e origine $\{S_0\}$: $T_0 = P_{vs} = \begin{bmatrix} x_{vs} & y_{vs} & z_{vs} \end{bmatrix}^T$
  (questa traslazione iniziale verrà continuamente convertita nella terna corrente S_i})

  Traslazione in terna $\{S_i\}$: $T_{si} = R_v^i \cdot T_0 = \begin{bmatrix} \cos(\psi_i) & \sin(\psi_i) & 0 \\ -\sin(\psi_i) & \cos(\psi_i) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{vs} \\ y_{vs} \\ z_{vs} \end{bmatrix} = \begin{bmatrix} x_{si} \\ y_{si} \\ z_{si} \end{bmatrix}$

  Matrice Omogenea : $H_{si} = \begin{bmatrix} \cos(\psi_i) & \sin(\psi_i) & 0 & -x_{si} \\ -\sin(\psi_i) & \cos(\psi_i) & 0 & -y_{si} \\ 0 & 0 & 1 & -z_{si} \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- Tralasciamo la fase di decollo in cui il `drone` a partire dalla sua origine iniziale prende quota lungo l'asse Z e supponiamo per semplicità di rappresentazione che non vi siano (per adesso) ulteriori rotazioni o traslazioni durante questa fase.

- In ogni istante il `Vicon` fornisce la posizione del `Drone` espressa nel sistema V inizialmente pari a P_{vs} ma in generale denominata con P_d^v che viene dunque riportata nel sistema S_i prima di inviarla come misura per l'aggiornamento del filtro di Kalman. In realtà l'invio a Kalman è fatto si in ogni istante come appena detto, ma soltanto per gli istanti successivi alla fase di decollo.

  Generica posizione del `Drone` espressa in `V`: $P_d^v = \begin{bmatrix} x_d & y_d & z_d \end{bmatrix}^T$

Costruiamo il vettore omogeneo : $P_{dh} = \begin{bmatrix} x_d & y_d & z_d & 1 \end{bmatrix}^T$

Lo moltiplichiamo per la matrice Omogena `H` : $P_{sh} = P_{dh} \cdot H = \begin{bmatrix} x_s & y_s & z_s & 1 \end{bmatrix}^T$

Lo riportiamo ad essere un vettore non omogeneo: $P_d^{si} = P_{dh} \cdot H = \begin{bmatrix} x_s & y_s & z_s \end{bmatrix}^T$
(Abbiamo quindi ottenuto il vettore posizione del drone non più espresso secondo la terna V ma secondo la terna S_i in quanto come detto in precedenza il drone è convinto di essere costantemente orientato come la terna $S_\#$ ed avere quindi angolo di yaw nullo. Se non avessimo proceduto alla conversione nella terna S_i non sarebbe stato in grado di effettuare tale conversione in autonomia.)

- Supponiamo che il drone sia decollato ad esempio ad una quota di $\mathbf{Z = 0.5\ m}$ e che la Wand si trovi in posizione P_{vf}^v (rispetto alla terna V). Ricordiamo che l'orientazione della `Wand` è del tutto irrilevante ai nostri scopi e pertanto verrà sempre trascurata. A questo punto il `Vicon` ci fornisce P_{vf}^v e noi la andiamo a ruotare in accordo con l'attuale orientazione della terna Body S_i per poi traslarla riconducendo l'origine a quella della terna `navigation` . In questo modo otteniamo come riferimento di posizione per il Drone il vettore P_{vf}^{si}.

  $P_{vf}^v = \begin{bmatrix} x_{vf} & y_{vf} & z_{vf} \end{bmatrix}^T$, costruisco il vettore omogeneo $P_{vfh} = \begin{bmatrix} x_{vf} & y_{vf} & z_{vf} & 1 \end{bmatrix}^T$
  ottengo infine : $P_{vf}^{si} = H \cdot P_{vfh} = \begin{bmatrix} x_{sf} & y_{sf} & z_{sf} & 1 \end{bmatrix}^T$

- A questo punto il nuovo riferimento è inviato al `Drone` che lo insegue. Durante il tragitto continuiamo a inviare, come prima, la sua posizione al filtro di `Kalman` continuando a fargli credere che il suo angolo di `yaw` è rimasto nullo. Possiamo farlo in quanto il riferimento che deve inseguire è già stato ruotato di conseguenza.

- Una volta che il `Drone` raggiunge la posizione desiderata, supponiamo che (a causa di eventuali disturbi durante il tragitto) abbia variato il suo angolo di `yaw` e che adesso sia pari a ψ_1 mostrato nella figura 25 relativamente all'istante t_1 . In questo caso infatti il sistema `Vicon` rileverà il nuovo valore dell'angolo che sarà utilizzato per ricalcolare la matrice omogenea come segue:

  Matrice di Rotazione da $\{S_1\}$ a $\{\mathrm{V}\}$: $R_{s1}^v = R_z(\psi_1) = \begin{bmatrix} \cos(\psi_1) & -\sin(\psi_1) & 0 \\ \sin(\psi_1) & \cos(\psi_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

  Matrice di Rotazione da $\{\mathrm{V}\}$ a $\{S_1\}$: $R_v^{s1} = (R_{s1}^v)^T = \begin{bmatrix} \cos(\psi_1) & \sin(\psi_1) & 0 \\ -\sin(\psi_1) & \cos(\psi_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Traslazione tra origine {V} e origine $\{S_0\}$: $T_0 = P_{vs} = \begin{bmatrix} x_{vs} & y_{vs} & z_{vs} \end{bmatrix}^T$

Traslazione ruotata in $\{S_1\}$: $T_{s1} = R_v^{s1} \cdot T_0 = \begin{bmatrix} \cos(\psi_1) & \sin(\psi_1) & 0 \\ -\sin(\psi_1) & \cos(\psi_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{vs} \\ y_{vs} \\ z_{vs} \end{bmatrix} = \begin{bmatrix} x_{s1} \\ y_{s1} \\ z_{s1} \end{bmatrix}$

Matrice Omogenea : $H_{s1} = \begin{bmatrix} \cos(\psi_1) & \sin(\psi_1) & 0 & -x_{s1} \\ -\sin(\psi_1) & \cos(\psi_1) & 0 & -y_{s1} \\ 0 & 0 & 1 & -z_{s1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- Adesso viene ricalcolata la nuova posizione della `Wand` che verrà riconvertita utilizzando la nuova matrice appena calcolata per fornirla al `Drone` come nuovo riferimento da inseguire. Allo stesso modo continuerà ad essere inviata al filtro di Kalman la misura di posizione attuale del `Drone` , anch'essa fornita in sistema V e riconvertita utilizzando la nuova matrice appena calcolata.

  Notiamo come già ripetuto ormai più volte in precedenza che per ogni variazione all'angolo di yaw il drone continua a ricevere riferimenti di posizione da inseguire e stime della sua posizione entrambi riferiti ad un sistema posizionato sulla terna `navigation` ma orientata come la corrente terna `body` in quanto continua ad essere convinto di avere yaw nullo e pertanto noi continuiamo a inviargli informazioni in cui la "correzione" dell'orientazione è già stata calcolata.

- Il tutto si ripete con una frequenza di 100 Hz fin quando la `Wand` non viene spenta.

- In questo istante il sistema `Vicon` fornirà come posizione della Wand il valore **[0, 0, 0]** che coincide con il valore fornito ogni qualvolta, per qualsiasi motivo, il `Vicon` non riesce a individuare la `Wand` in uno o più frame. Avendo dunque deciso di interpretare questo particolare valore di posizione come codice di errore (in quanto considerando il vincolo del pavimento e la presenza di rumore nelle misure è assolutamente improbabile se non impossibile che un oggetto si trovi in quella posizione esatta per uno o addirittura più istanti consecutivi) nel caso in cui questo venga ricevuto noi continuiamo a mantenere il drone fermo nella sua ultima posizione e, qualora il valore sia fornito per un dato numero di istanti consecutivi, decidiamo di intraprendere la fase di atterraggio per concludere quindi l'esperimento.

Fasi di Test

Detto quanto basta per poter comprendere il codice che siamo andati a scrivere, elenchiamo ora le varie fasi di test che abbiamo attraversato per arrivare all'esperimento finale. Le fasi sono tra loro progressive. Per alcuni test è disponibile sia il relativo file omonimo (.py), sia un video omonimo (.mp4) in cui si mostra il risultato di ciascuna esecuzione.

TEST_1.py:
In questo file abbiamo testato le funzioni della libreria py-vicon e della cf-lib che meglio riuscissero a far si che il Drone, dopo una iniziale fase di decollo, iniziasse a inseguire la posizione della Wand. In questo caso il nuovo riferimento di posizione non era dato ad ogni istante ma ad intervalli di tempo prefissati. All'interno di ogni intervallo il riferimento era mantenuto costante e pari all'ultimo inviato. Il risultato è un **"inseguimento a tratti"** della Wand da parte del Drone.

Il Drone deve essere posizionato esattamente al centro della stanza e orientato esattamente come la terna fissa "V". Soltanto dopo essersi assicurati che sia così può essere acceso e può essere eseguito l'esperimento.

TEST_2.py:
A questo punto abbiamo eliminato il vincolo relativo al riferimento costante all'interno degli intervalli di tempo ed abbiamo iniziato a inviare in ogni istante (ad una frequenza di 100 Hz) la posizione corrente della Wand. Si ha dunque un "**inseguimento in tempo reale**".

TEST_3.py:
Dato che fino a questo momento l'esperimento non aveva un vero "**criterio di arresto**", abbiamo deciso di far interrompere l'esperimento facendo atterrare gradualmente il Drone a partire dalla posizione corrente nel momento in cui la Wand viene spenta. La logica adottata è la stessa già spiegata in precedenza per cui al momento dello spegnimento della Wand si ha l'invio da parte del Vicon della particolare posizione della Wand pari a [0, 0, 0] che abbiamo interpretato come codice di errore e che, qualora fosse fornita per diversi istanti consecutivi da logo all'inizio della procedura di atterraggio.

TEST_4.py:
Qui è dove abbiamo eliminato il "vincolo" di dover iniziare l'esperimento facendo partire il Drone dall'origine del sistema Vicon e con orientazione relativa rispetto alla terna "V" nulla. Per fare ciò abbiamo utilizzato la posizione e orientazione del Drone rispetto alla terna Vicon per creare le relative matrici omogenee di rototraslazione che permettessero di ricevere informazioni in terna Vicon e inviare riferimenti al Drone in terna Body in modo del tutto coerente secondo quanto spiegato in precedenza.

Il Drone può quindi essere **inizialmente posizionato ed orientato in modo del tutto arbitrario** (purchè parta ovviamente sempre dal suolo e senza pendenze).

File Definitivo

Inseguimento

`INSEGUIMENTO.py` costituisce il file definitivo ed è quello in cui abbiamo aggiunto dei flag in modo da ottenere un unico file in cui a seconda del loro valore (impostato prima dell'esecuzione e eventualmente modificato tra esecuzioni successive) otteniamo l'esecuzione di diversi rami del codice e che portano dunque all'esecuzione dell'esemperimento in diverse modalità:

- `MAKE_LOG_FILE:`
  Se questo flag è posto ad 1 l'esecuzione dell'esperimento produrrà un file di log in cui saranno salvate tutte le informazioni di interesse (riferimenti inviati al drone, misure di posizione del drone inviate al filtro di Kalman, posizione e orientazione fornite da Vicon...).

- `KALMAN_INCLUDE_QUATERNION:`
  Se questo flag è posto ad 1 l'esecuzione prevederà che il drone riceva dal sistema Vicon oltre che all'informazione di posizione anche quella di orientamento (fornita per mezzo di quaternioni).

- `ACTIVATE_KALMAN_DURING_TAKEOFF:`
  Se questo flag è posto ad 1 l'esperimento avrà luogo in modo che anche durante il decollo il drone riceva informazioni per l'aggiornamento del filtro di Kalman (posizione o posa a seconda del valore del precedente flag).

- `LOG_TEST_WITH_DISACTIVATED_THRUSTER:`
  Questo falg se abilitato consente di condirre esperimenti a mano libera, senza bisogno della wand e a "motori spenti" in modo da poter muovere il drone nello spazio semplicemente prendendolo in mano e guidandolo dove vogliamo. Durante questo tipo di esecuzione le informazioni raccolte dipenderanno dai valori assegnati ai flag precedenti.

Facciamo presente che tutta la parte di relazione finora descritta è riferita al caso in cui i valori dei flag sono rispettivamente:

- `MAKE_LOG_FILE:1`
- `KALMAN_INCLUDE_QUATERNION:0`

- `ACTIVATE_KALMAN_DURING_TAKEOFF:0`
- `LOG_TEST_WITH_DISACTIVATED_THRUSTER:0`

Facciamo presente che dal codice si nota una prima sezione in cui oltre ai flag appena spiegati è possibile modificare quelle variabili che, a seconda di dove l'esperimento è condotto o di chi lo sta conducendo, potrebbero assumere valori diversi e che pertanto abbiamo reso "parametrici" in modo che possano essere semplicemente modificate una sola volta prima dell'esecuzione senza rendere necessarie modifiche interne al codice vero e proprio. A titolo di esempio, stiamo parlando di variabili come "nome associato agli oggetti nel sistema Vicon", "URL della connessione tra Crazyradio e Crazyflie", "IP e porta di connessione relative al sistema Vicon"...

Relativo

In aggiunta al file `INSEGUIMENTO.py` abbiamo fornito un file analogo ma denominato `RELATIVO.py` e per cui vale tutto quanto detto finora ad eccezione del fatto che l'esecuzione non prevede l'inseguimento del moto della Wand ma il Drone, una volta decollato, a partire dalla sua posizione corrente insegue soltanto il moto relativo della Wand. Ad esempio, se la Wand "disegnerà" nello spazio un quadrato a partire dalla sua posizione iniziale, il Drone, a partire anch'esso dalla sua posizione iniziale (diversa ovviamente da quella della Wand) replicherà nello spazio il "disegno" del quadrato fatto dalla Wand. Il tutto in tempo reale ad una frequenza di 100 Hz.
La trattazione di quest'ultima modalità non è riportata in quanto si basa esattamente sugli stessi concetti spiegati finora e validi per la prima modalità; l'unica differenza è costituita appunto dal riferimento di posizione inviato al drone che non è più coincidente con la nuova posizione della Wand ma diventa coincidente alla posizione attuale del Drone a cui viene sommata la **traslazione** che la Wand ha compiuto rispetto alla sua ultima posizione.

Interfaccia

Come ulteriore possibile sviluppo forniamo un file `INTERFACCIA.py` in cui abbiamo implentato una prima semplicissima interfaccia da cui rendiamo possibile consultare un file README.txt (riportato nell'ultima sezione di questa relazione) in cui sono contenute le istruzioni su come poter essere in grado (come noi) a partire da zero di poter eseguire l'esperimento. Qui troviamo le indicazioni su come si deve predisporre l'ambiente, quali librerie dover installare, quali sono le osservazioni più importanti e la procedura da dover seguire per la corretta riuscita dell'esperimento. Da questa semplice interfaccia è inoltre possibile cliccare su un pulsante `START` ed eseguire quindi uno soltanto dei due file sopra descritti (`INSEGUIMENTO` , `RELATIVO`). Precisiamo che, se eseguito così come è scritto, il file `INTERFACCIA.py` eseguirà soltanto un banale esempio di prova in cui viene calcolato randomicamente un numero intero per poi stamparlo nel form dopo

un ciclo di attesa di molte iterazioni (questo per testare il fatto che click consecutivi successivi al primo sul pulsante start sono disabilitati dirante l'esecuzione del codice lanciato) .
Qualora si voglia utilizzare il file per eseguire l'esperimento vero e proprio si prega di leggere attentamente i vari commenti presenti tra le righe di codice e di seguire la procedura riportata nel file README.txt.

Possibili Sviluppi

Volendo pensare a possibili sviluppi futuri, le nostre idee sono state le seguenti:

- Sicuramente come prima cosa dovrebbe essere trovato un modo per poter inviare al Drone anche le misure di orientazione senza inficiare sulle performance del volo. Attualmente infatti volendo inviare al filtro di Kalman del Drone anche l'informazione dell'orientazione contenuta nei quaternioni (per poter quindi evitare di "pre-ruotare" le informazioni inviate e lasciare che il Drone stesso effettui questa correzione) otteniamo come risultato una instabilità durante il volo che ne causa la caduta immediata.
  Tra gli elementi emersi durante l'analisi di questo problema e classificati come "possibili cause" facciamo presente che, come si può evincere plottando i file di log (presenti nella cartella /LOG) i valori dell'angolo di Pitch forniti dal Vicon e ottenuti dalla tabella di log del Drone risultano essere di segno opposto; inoltre, sempre dal confronto tra i valori forniti dal Vicon e quelli ottenuti dalla tabella di log del Drone, emergono vari ritardi più o meno marcati a seconda delle condizioni in cui si effettuano gli esperimenti.

- In secondo luogo si può pensare di estendere l'interfaccia in modo che contenga più pulsanti al fine di poter lanciare indifferentemente una tra le due versioni disponibili (inseguimento del moto assoluto e relativo) senza bisogno di attuare ogni volta la procedura di modifica dei nomi dei relativi file e dei relativi codici.

- Una volta implementati i punti precedenti si può pensare di replicare quanto appreso da questo esperimento per riprodurne uno che utilizzi non un solo drone ma una formazione.

README.txt

REQUIREMENTS:

1)
VICON SYSTEM:
- TRACKER SOFTWARE
- ACTIVE WAND

2)
BITCRAZE HARDWARE:
- CRAZYFLIE
- FLOWDECK
- CRAZYRADIO

3)
WINDOWS PC (Not the same PC where the Tracker runs)

4)
Python 3.7 Version is HIGHLY RECOMMENDED. We invite you to
delete all other versions in order to avoid conflicts.

5)
Matlab version 2020 in order to plot the results from the log
file. The file you have to open is called "plot_log_file.m"

INSTALLATION OF THE REQUIRED LIBRARIES:

Install numpy:
From the Prompt go to the directory
"C:\Users\'your username'\AppData\Local\Programs\Python\Python37\Scripts"
Type the command "pip install numpy"

Install Vicon Datastream_SDK:
Double-Click on "install_vicon_dssdk" to install.
Delete file "pip".

```
Install USB:
From the Prompt go to the directory
"C:\Users\'your username'\AppData\Local\Programs\Python\Python37\Scripts"
Type the command "pip install pyusb"
Type the command "pip install libusb1"

Install threading:
From the Prompt go to the directory
"C:\Users\'your username'\AppData\Local\Programs\Python\Python37\Scripts"
Type the command "pip install threaded"

Install CrazyRadio Driver:
Connect the CrazyRadio to a USB port
Go to the link: "https://zadig.akeo.ie/" and download the last version
Use this link as Guide for installation:
    "https://wiki.bitcraze.io/doc:crazyradio:install_windows_zadig"



INTERFACE.py

If you want to use the "Interface.py" please search for the
latest version of the QtCreator package and download it.

You also have to :
a)
Chose one of the files : "INSEGUIMENTO.py" , "RELATIVO.py" and
change its name in "interface_main_file.py"


b)
Crate a new function where you "transfer" all the code. You can
call that function "Imperio(Dialog)" if you don't want to
change anything else in the code.
```

OBSERVATIONS:

0)
Before running the code it is highly recommended to read all
comments in the file ! In that comments in fact we explained
also everything about all the values you have to set before
launching the experiment.

1)
During the execution we continuously receive the position of
the Wand from the Vicon Tracker and when we want to interrupt
the experiment we have to turn off the wand.
When this happens, the function "GetSegmentGlobalTranslation(..)"
used for receiving the Wand Position from the Vivocn tracker
doesn't return an exception but it just returns the position
values: [0,0,0]. Taking into consideration the fact that it is
highly improbable that the Wand will have this exact position
for more than two consecutive frames, we decide to consider
"MAX_LOSS" consecutive Wand's position with value [0,0,0] as
the error encoding for the end of the experiment.

PROCEDURE FOR THE EXECUTION OF THE EXPERIMENT:

0)
It is highly recommended to start a new calibration of the
cameras for each session.

1)
Place markers on both Drone and Wand in order to crate the
relative objects with the Vicon Tracker.

2)
Set the body frame for the Drone-Object on the Vicon in order
to have the same orientation of the body frame of the Drone.

3)
Check that both Wand and Drone are correctly seen by the Vicon Tracker. Otherwise try a different placing for the markers. Check also that the body frame for the drone object has this axis orientation:
- Z up
- X stern-bow
- Y right-left watching form stern to bow

4)
Choose a starting point for the Drone and place it on the floor of the flight room. Only when you placed the drone on the floor you can turn it on. It is important to maintain the drone placed on the floor with no pitch and no roll angles also WHILE you press the button for turning it on, this is because when you press the button the drone considers its current orientation and position as initial values for its body frame's origin and orientation so, if you don't follow this procedure, it will consider wrong values of initial orientation. Regarding to the position instead, when you turn on the drone it will consider the current position as the origin of its body frame so you don't have to change its position once you pressed the start button.

5)
Once both Drone and Wand are turned on you can start the experiment pressing the START button of the interface.
If you don't use the "Interface.py" you can simply run your main file.

6)
The experiment will be concluded when you will turn off the ActiveWand. If something wrong happens and you want to analyse it critically, we suggest to set = 1 the flag "MAKE_LOG_FILE" so at the end of the experiment you will find all information you need.

7)
Every time that the experiment ends (correctly or not) it is
very important to turn off the Drone and then turn it on again
from the new starting position of the new experimentation.
In this way you can be sure that both orientation and position
are correctly initialised.

8)
Enjoy the flight !