INTERNAL REPORT

SRT Photogrammetry Simulator

Andrea Saba

Report N. 48, released: 28/05/2015

Reviewer: Marco Buttu

OAC Osservatorio Astronomico di Cagliari

SRT Photogrammetry Simulator

Andrea Saba

28 maggio 2015

Indice

Sommario

Il Sardinia Radio Telescope (SRT) è un radiotelescopio realizzato in configurazione gregoriana costituito da uno specchio primario di 64 metri di diametro, di profilo quasi parabolico, e da uno specchio secondario detto anche subriflettore, quasi ellittico. SRT lavorerà a frequenze comprese tra 300 MHz e 100 GHz, corrispondenti a lunghezze d'onda comprese tra 1 m e 3 mm. Lo studio delle deformazioni della struttura e delle ottiche SRT riveste un ruolo fondamentale per permettere l'operatività di SRT ad alta frequenza, in quanto l'efficienza di un radio telescopio è strettamente legata alla deviazione delle ottiche delle loro superfici ideali.

Essendo dotato SRT di una superficie attiva dello specchio primario ed essendo questa superficie soggetta a deformazioni dovute alla forza di gravità, a quella del vento e alle dilatazioni termiche, è necessario correggere la forma della superficie primaria per fare in modo che abbia sempre la forma ideale. Per verificare che la forma della superficie si avvicini il più possibile a quella ideale si utilizzano tecniche di metrologia. Una di queste è la fotogrammetria che, basata sull'identificazione dello stesso punto del mondo reale su diverse prese fotografiche realizzate da diversi punti di vista, permette la ricostruzione della posizione reale dei punti nello spazio con una accuratezza sufficiente per caratterizzare l'antenna.

Nell'ambito del progetto di ricerca "Studio delle deformazioni di SRT mediante fotogrammetria e tecniche di computer vision." finanziato dalla Regione Autonoma della Sardegna, si studia un piano di fattibilità dell'utilizzo della fotogrammetria per la correzione in tempo reale della deformazione della superficie primaria del Sardia Radio Telescope migliorando l'efficienza dell'antenna nel range più alto di frequenze operative.

Nel presente documento verrà illustrata la realizzazione di un ambiente di simulazione di prese fotogrammetriche.

Il simulatore si compone di 3 parti:

1. Generatore di targets fotogrammetrici codificati e non
2. Ambiente di modellazione 3D per la ricostruzione della scena e la realizzazione delle prese sintetiche
3. Software per il bundle adjustment

Il bundle adjustment è un procedimento tramite il quale dato un insieme di immagini in cui sono stati proiettati dei punti dello spazio 3D si è grado di calcolare, con il minimo errore possibile, la reale posizione iniziale dei punti nello spazio 3D.

L'obbiettivo è di determinare il numero di prese minimo necessario e le coordinate dei punti di presa. Queste ultime infatti devono essere tali che ogni singola posizione della camera da presa sintetica sia nel mondo reale una posizione favorevole. Un problema che in questa situazione non può essere affrontato empiricamente per problematiche sia pratiche che economiche.

La soluzione più efficace è appunto la costruzione di un ambiente di simulazione per poter verificare e determinare a priori il maggior numero di variabili del problema.

Per valutare la precisione dell'ambiente di simulazione sono infine state messe a confronto le coordinate utilizzare dall'ambiente di modellazione 3D per il posizionamento dei targets con le coordinate degli stessi punti restituite dal bundle adjustement.

1 Fotogrammetria

La fotogrammetria è una tecnica che consente la rilevazione di informazioni metriche, forma e dimensione di oggetti tridimensionali a partire da immagini fotografiche, realizzate da diversi punti di vista. Questa tecnica si basa sulle equazioni di collinearità.

Le condizioni (o equazioni) di collinearità rappresentano le relazioni fra le coordinate assolute (x', y', z') del punto immagine P', le coordinate (X_P, Y_P, Z_P) del corrispondente punto oggetto P e le coordinate (Z_0, Y_0, X_0) del punto di presa O.

La condizione che queste esprimono è che, nel caso ideale, al momento dello scatto (presa) punto oggetto, centro di presa e punto immagine risultano allineati lungo una retta.

Le trasformazioni espresse dalle equazioni di collinearità definiscono la prospettiva centrale di un oggetto tridimensionale che è definita, per ogni fotogramma, da 9 parametri indipendenti:

- 3 di orientamento interno
  - c distanza principale o distanza focale (costante della camera)
  - x_0, y_0, coordinate del punto principale
- 6 di orientamento esterno
  - X_0, Y_0, Z_0 coordinate assolute del centro di presa O
  - w, f, k, tre rotazioni d'assetto (contenute nei elementi a_{ij} della matrice di rotazione)

$$\begin{cases} X_P = X_1^0 + (Z_P - Z_1^0)\frac{a_{11}x' + a_{12}y' + a_{13}z'}{a_{31}x' + a_{32}y' + a_{33}z'} \\ Y_P = Y_1^0 + (Z_P - Z_1^0)\frac{a_{21}x' + a_{22}y' + a_{23}z'}{a_{31}x' + a_{32}y' + a_{33}z'} \end{cases} \tag{1}$$

Per trovare i valori di queste incognite è necessario linearizzare un sistema di equazioni di collinearità basato sullo stesso punto reale proiettato su diverse prese fotografiche. Questo permette di calcolare i parametri interni ed esterni del sistema e quindi ottenere la ricostruzione della scena reale. Per rendere automatizzata la correlazione degli stessi punti su più prese fotografiche vengono utilizzi degli oggetti (target) con una sagoma o una figura note a priori e, eventualmente, corredati di uno schema visivo univoco chiamato codifica.

Le singole prese possono contenere o tutto l'oggetto preso in considerazione o una porzione di questo con la condizione che la distanza sia tale che i punti di riferimento all'interno della stessa immagine siano presenti in numero sufficiente da poter identificare univocamente la posizione di presa, quindi l'orientamento della macchina da presa.

Le diverse prese devono essere in relazione tra loro in modo che i punti di interesse siano presenti su diverse prese e garantiscano una sovrapposizione parziale delle singole coppie di prese.

Nell'ipotesi di non avere vincoli nel posizionare la camera da presa e non avendo impedimenti visivi quali ostacoli o illuminazione non ottimale, il problema si traduce nella sola elaborazione delle immagini.

Nel nostro caso specifico le prese della superficie primaria saranno legate a diversi vincoli, alcuni strutturali, come le zone d'ombra create del tetrapode e dalla camera del gregoriano; altri logistici non essendo possibile posizionare la camera in tutti i punti dello spazio. A questo si aggiunge il vincolo legato all'ipotesi di valutare l'installazione delle camere fotogrammetriche in maniera solidale alla struttura di SRT.

2 Ambiente di sviluppo

Come illustrato in precedenza le tre componenti del simulatore sono: il generatore di targets, l'ambiente di modellazione 3D e il software di bundle adjustement.

Il generatore di targets scritto in Python, si occupa di creare immagini del target che verranno posizionate sulla scena dell'ambiente di modellazione, come verrà illustrato nel dettaglio più avanti. L'ambiente di modellazione 3D è Blender[1] che permette di sfruttare il motore interno di rendering per la realizzazione delle prese fotografiche sintetiche rispettando il più possibile le caratteristiche ottiche di una macchina fotogrammetrica modellata. Blender permette inoltre tramite il sul motore di scripting di scrivere in Python le automazioni che posizionano nello spazio il modello del radiotelescopio, i targets sulla superficie primaria e le camere da presa. Tramite le camere da presa vengono quindi create le prese sintentiche della scena.

Il software scelto nel nostro caso per il bundle adjustement è AICON3D della AICON 3D system[5] di cui l'Osservatorio Astronomico di Cagliari è in possesso di una licenza.

3 Generatore di targets

In fotogrammetria i targets sono oggetti che all'interno delle prese fotografiche possono essere riconosciuti in maniera precisa nella loro posizione permettendo la più bassa ambiguità possibile con gli altri elementi dell'immagine. Possono essere degli oggetti bidimensionale o tridimensionali e a seconda delle esigenze di presa possono essere opachi, rifrangenti o generare luce propria. Caratteristica fondamentale è che la loro posizione sia solidale all'oggetto.

Prendendo in considerazione i targets bidimensionali senza luce propria, questi possono essere di diversa forma, ma in generale rispettano la condizione di presentare un alto contrasto e la precisa individuazione del centro dello stesso. Sono preferite delle forme con simmetria centrale come cerchi, quadrati, rombi, cerchi concentrici e stelle. Queste forme permettono un riconoscimento tramite template matching con software di image processing.

Per rendere automatico il processo di correlazione tra i targets individuati in diverse prese, questi possono essere corredati da una codifica visiva presente in prossimità del target (circonferenze concentriche, porzioni di arco di cerchio, ...), o che costituiscono il target stesso (dispersione di cerchi, matrice di punti, ...)

Questa codifica può essere di diversi tipi a seconda delle condizioni ambientali e della distanza, ma in ogni caso deve permettere una individuazione precisa del centro del target indipendentemente dall'angolo di presa.

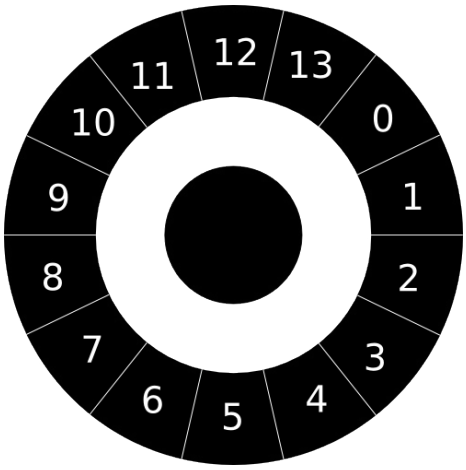

Figura 1: Codifica a 14bit

Il generatore di targets è un'applicazione scritta in Python 2.7 che utilizza le librerie di terze parti `cairo` e `pyPdf` e il modulo `math` della libreria standard.

`Cairo` è una libreria grafica sotto licenza LGPL per la grafica vettoriale che permette la creazione di files in diversi formati. Nel programma la libreria è utilizzata per generare files in formato SVG, PS e PDF e per esportare in formato PNG.

La libreria `pyPdf` è utilizzata invece per unire i documenti in formato pdf in un unico file.

La libreria `math` è utilizzata per avere un calcolo preciso degli archi di cerchio della codifica dei targets.

3.1 Targets specifici

Nel nostro caso sono stati presi in considerazione i targets costituiti da un cerchio nero su sfondo bianco o bianco su sfondo nero in cui il colore bianco, nel caso di targets rifrangenti, e costituito da materiale rifrangente.

La loro versione codificata è identificabile dalla presenza o meno di porzioni di arco di un anello che circonda il target secondo una rappresentazione binaria. Ogni porzione di arco di cerchio ha la stessa lunghezza.

Per convenzione da ora in seguito assoceremo la presenza di un arco al simbolo 1 e la sua assenza al simbolo 0. Il verso di lettura sarà orario e partirà dalla posizione 0 indicata nelle due figure successive.

Viene presa in considerazione la codifica a 14 bit riconosciuta dal software Aicon3D Studio 10.06.06 della Aicon 3D System.

3.2 Definizione della codifica a 14 bit

In questa codifica il numero di archi è pari a 14 come si può vedere in Fig. **??**; non accetta tutte le possibili combinazioni di presenza o assenza di archi di cerchio ma accetta solo quelle che rispettano le seguenti condizioni:

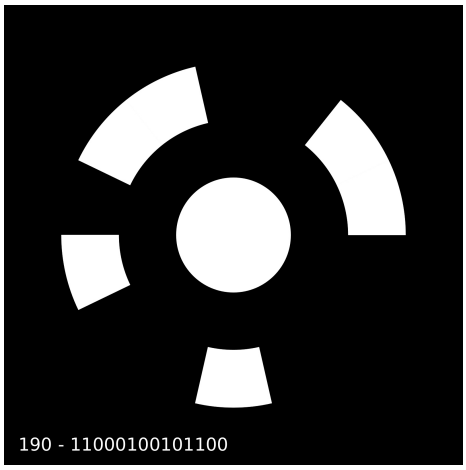

Figura 2: Esempio di codifica

1. la posizione 0 viene gestita con un bit di parità ed è presente solo per fare in modo che il numero totale di archi di cerchio posti ad 1 sia pari
2. il bit/arco più significativo è il numero 1 quello meno significativo il numero 13
3. il numero totale di bit/arco posti ad 1 deve essere maggiore o uguale a 4
4. non possono esistere due codifiche valide che siano ottenibili dalla rotazione dell'una nell'altra.
5. Devono sempre essere presenti almeno due bit/arco posti a 1 che si trovano in posizione opposta rispetto al centro.
6. Il primo codice valido è etichettato come numero 2 ed ha la codifica

`11000000100001`

7. I codici successivi sono etichettati come 3, 4, 5, etc. e si ottengono aggiungendo un 1 alla codifica precedente e ponendo ad 1, se necessario (punto 1), il bit di parità in posizione 0

Alcuni codici con etichette superiori a 299 sono dedicati ad utilizzi speciali.

3.3 Il modulo per la generazione dei targets

Per poter eseguire correttamente il codice è necessario, oltre ad una versione corretta dell'interprete Python, aver installato le librerie `cairo`, e `pyPdf`.

Nel modulo sono presenti le seguenti funzioni:

```
codetostring(code, bitcode=14):

checkrotationcode(dic, newcode, newCodestring_count1,
    bitcode=14):
```

```
createnext(dic, currentCode, bitcode=14, rand =False):

returncodingSigma3D(bitcode=14, rand =False):

createsingletarget(targetnumber, originalCodestring,
    targetcode, currentCodestring, bitcode, ctx, ctx_ps
    , ctx_pdf, createPS, createPDF, targetsize =
    34.01553376, targetsizeschema = [5,5,5,5],
    background = 1):

createtarget(targetnumber, targetcode, bitcode=14,
    surfaceSVG=None, surfacePS=None, surfacePDF=None,
    createPS = True, createPDF=True, WIDTH = 1600,
    HEIGHT = 1600, targetsize = 800, targetsizeschema =
     [5,5,5,5], page = 0, row = 0, column = 0, padding
    = 0, background = 1):

createtarget_by_dictionary(dic, bitcode=14, folder =
    None, pagesize = "A4", rows = 1, columns = 1,
    targetsize = None, targetsizeschema = [5,5,5,5],
    createPS = True, createPDF=True, padding = 0,
    background = 1, PNGOnly = False, UncodedTargets =
    False):
```

Le funzioni `codetostring`, `checkrotationcode`, `createnext` e `returncodingSigma3D`, si occupano della generazione della lista di codici validi con le relative etichette; mentre le restanti funzioni si occupano delle creazione grafica dei targets e della loro impaginazione.

3.4 Generazione dei codici

La funzione `returncodingSigma3D` prende in input il numero di bit della codifica e restituisce un dictionary i cui elementi sono l'etichetta del target e il valore della sua codifica binaria. I valori validi ammessi dal parametro bitcode sono 14, e 20 ma solo i risultati generati dal valore 14 sono stati verificati. La funzione `codetostring` prende in input il valore della codifica di un target e il numero di bit della sua codifica e restituisce una stringa di 0 ed 1 corrispondente alla sua rappresentazione binaria.

La funzione `createnext` prende in input il dictionary dei codici validi creati, il successivo codice candidato e il numero di bit della codifica. Questa funzione in particolare si occupa della verifica delle condizioni 3 e 5 viste in precedenza nel caso della codifica a 14 bit.

La funzione `checkrotationcode` prende in input gli stessi parametri della funzione precedente e restituisce vero o falso a seconda del caso in cui sia verificata o meno la condizione 4 della codifica a 14 bit vista prima.

3.5 Creazione dei targets e impaginazione

La funzione `createtarget_by_dictionary` è l'interfaccia del modulo e i suoi parametri permettono di definire tutti gli aspetti della creazione e dell'impagi-

nazione dei targets.

Nel dettaglio i parametri hanno la seguente funzione

dic è il dictionary dei codici che si vogliono creare, composto dall'etichetta del codice e il suo valore. In generale questo dictionary è quello restituito in output dalla funzione `returncodingSigma3D`

bitcode è il numero di bit della codifica che è stata utilizzata nel dictionary del parametro `dic`. Deve essere indicata perché nel dictionary sono presenti i valori dei codici ma non il numero di bit della rappresentazione binaria.

folder indica il nome della sotto-cartella in cui verranno salvati tutti i files di output. Se la cartella è già presente o viene passato il valore `None` i files verranno salvati nella posizione corrente.

pagesize indica la dimensione della pagina in cui verranno raggruppati i targets. Può assumere una stringa corrispondente alla serie A dello standard ISO 216[4]. Le stringhe valide sono "`A1`", "`A2`", "`A3`", "`A4`". La pagina è sempre verticale. Se viene indicata una stringa non valida la dimensione del foglio verrà impostata in `1600 x 1600` mm

rows numero di targets per riga della pagina

columns numero di targets per colonna della pagina

targetsize dimensione in mm del cerchio al centro del target. Se questo valore viene indicato verranno ignorati i parametri rows e columns che saranno ricalcolati in base allo spazio disponibile nella pagina. Se il valore non viene indicato la dimensione del target sarà adattata al numeri di targets per pagina rispettando i valori rows e columns.

targetsizeschema lista di valori interi che indicano le proporzioni tra il cerchio centrale del target e il suo anello codificato.

La lista è composta da 4 valori la cui somma deve essere 20. Il primo valore indica il raggio del cerchio del target. Il secondo valore indica il raggio della circonferenza interna dell'anello e il terzo il raggio della circonferenza esterna dell'anello. L'ultimo valore indica la distanza tra la circonferenza esterna dell'anello del target e il bordo dello sfondo del target. I valori standard sono `[5, 5, 5, 5]` per la codifica a 14 bit e `[4, 8, 4, 4]` per la codifica a 20 bit.

createPS impostato a `True` o `False` nel caso in cui si voglia o meno l'output di ogni pagina anche in formato PS

createPDF impostato a `True` o `False` nel caso in cui si voglia o meno l'output di ogni pagina anche in formato PDF. In questo caso verrà generato anche un file PDF con all'interno tutte le pagine di targets creati.

padding distanza in mm tra i targets presenti per pagina

background posto ad `1` indica che verranno creati dei targets bianchi su sfondo nero, posto a zero indica che verranno creati dei targets neri su sfondo bianco.

PNGOnly se impostato a `True` verranno ignorati i valori di `createPS` e `createPDF` e verrà generato solo un file PNG per ogni pagina di targets. Non verranno generati i files in formato SVG

UncodedTargets posto a `True` indica che si creerà oltre le pagine dei targets codificati una pagina di targets non codificati disposti in 4 per target

La funzione crea (tranne nel caso dell'utilizzo dell'opzione `PNGOnly`) ogni pagina anche in formato SVG, per permettere, se necessario, la modifica a posteriori direttamente da un formato vettoriale.

La funzione `createtarget` calcola la posizione di ogni target nella pagina corrente e chiama la funzione `createsingletarget` che disegna il target eseguendo i singoli comandi grafici.

3.6 Utilizzo della funzione `createtarget_by_dictionary`

Di seguito alcuni esempi di utilizzo della funzione `createtarget_by_dictionary`.

Per creare una serie di targets a 14 bit bianchi su sfondo nero, compreso il target non codificato, in cui la dimensione del target è di 2,00 cm divisi in pagine di formato A4 separati ognuno di 5,00 mm, compresi i formati PS, PDF, SVG, PNG, nella cartella "`test_A4_20mm_14bit_BN`".

```
createtarget_by_dictionary(returncodingSigma3D(bitcode
    =14), bitcode=14, folder = "test_A4_20mm_14bit_BN",
     pagesize = "A4", rows = 0, columns = 0, targetsize
     = 20, padding = 5, background = 0, UncodedTargets
    = True)
```

Per creare una serie di targets a 14 bit neri su sfondo bianco, senza il target non codificato, 3 per 4 targets per pagina di formato A3 separati ognuno di 5,00 mm, compresi i formati PDF, SVG, PNG, nella cartella "`test_A3_3x4_14bit_NB`".

```
createtarget_by_dictionary(returncodingSigma3D(bitcode
    =14), bitcode=14, folder = "test_A3_3x4_14bit_NB",
    pagesize = "A4", rows = 4, columns = 3, targetsize
    = None, padding = 5, background = 1, UncodedTargets
     = False, createPS = False)
```

4 Blender Addon

Per l'ambiente di Blender è stato scritto un Addon che permette:

- Il posizionamento nello spazio 3D dei targets codificati e non, a partire da un file di testo con le coordinate degli stessi, e un file con il profilo ideale della superficie primaria del Radiotelescopio.
- Il posizionamento delle camere fotogrammetriche sintetiche a partire da un file di testo che comprende il punto di presa e l'orientamento esterno delle stesse.
- Il posizionamento del modello 3D di SRT a partire da file in formato ply[2].

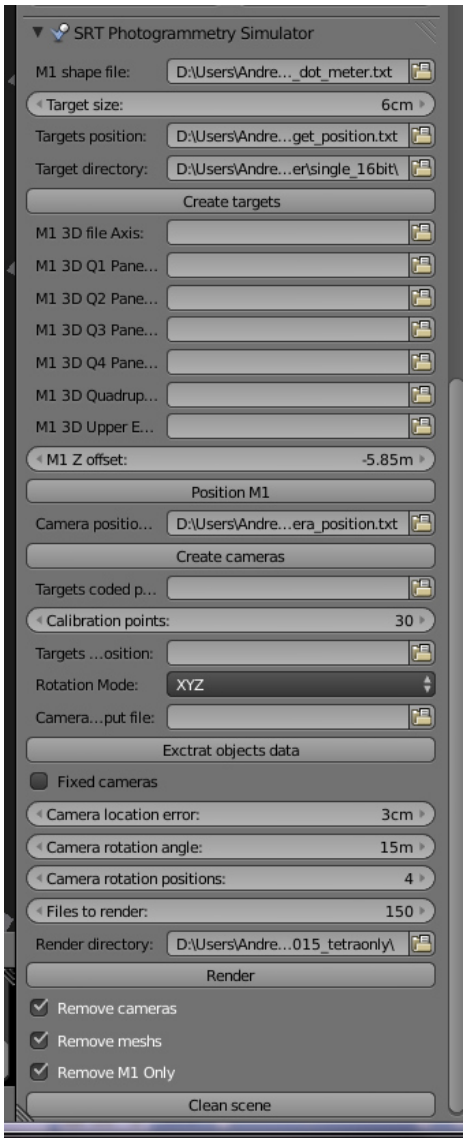

Figura 3: Intefraccia dell'Addon per Blender

L'addon inoltre permette di realizzare le prese sintetiche di tutte le camere. In particolare è stato previsto un movimento basculante delle camere lungo due assi. Per questo basculamento è possibile impostare l'ampiezza in gradi e il numero di spostamenti.

E possibile estrarre le informazioni relative alla posizione dei targets ed estrarre un numero arbitrario di questi per creare i ground control points che sono necessari ad AICON 3D Studio per effettuare le operazione di misurazione.

L'addon (Fig. 3) è stata suddivisa in 6 sezioni:

1. Creazione dei targets
2. Posizionamento del modello 3D di M1
3. Creazione delle camere
4. Salvataggio su file delle informazioni sulla posizione dei targets
5. Creazione delle prese sintetiche
6. Rimozione della scena degli oggetti presenti

4.1 Creazione dei targets

Per la creazione e il posizionamento dei targets sono necessari:

- un file di testo con il profilo delle parabola sul piano YZ
- la dimensione dei targets
- un file di testo con la posizione dei targets
- una directory in cui sono presenti le immagini dei targets

Il file di testo con il profilo della parabola è costituito dalle coordinate dei singoli punti sul piano YZ espresse in metri. Il file ha il seguente formato:

```
0.00000 0.000000000      0.000000000
0.00000 0.006350000      0.000000000
0.00000 0.012700000      0.000002540
0.00000 0.019050000      0.000002540
```

Il file non ha riga di intestazione né etichette identificative delle righe. I valori devono essere crescenti lungo la coordinata Y. La prima colonna non viene utilizzata dal simulatore. L'origine del sistema di riferimento è il vertice della parabola.

Nel nostro caso specifico viene utilizzata una curva ottenuta tramite laser-scanner chiamata generatrice. Questa generatrice rappresenta quindi l'approssimazione più fedele alla superficie reale, è costituita da circa 5000 coordinate e ha come centro l'origine del sistema di riferimento.

La dimensione dei targets è espressa in metri e il suo valore predefinito è 0,06 e rappresenta la dimensione del cerchio presente al centro del target. Il file con la posizione dei targets ha il seguente formato:

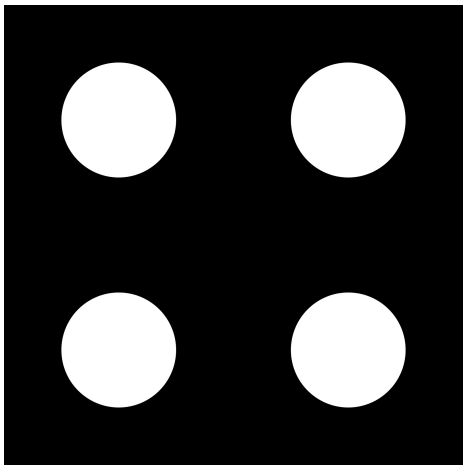

Figura 4: Target non codificato

```
%NODE            X                Y
           Z
60016            4.4570227        0.586761543
   6.04058739
60018            3.5665121        2.73666437
           6.04058739
60020            1.7203516        4.15327953
           6.04058739
```

dove la prima riga è una riga di intestazione e la prima colonna è un numero identificativo non utilizzato dal simulatore. Le tre coordinate indicate hanno origine coincidente con quella della generatrice della parabola. Solo le prime due coordinate vengono utilizzate mentre l'ultima viene calcolata dal simulatore tramite il file del profilo della parabola. L'ordine delle posizioni dei targets è ininfluente per la loro creazione, ma verrà utilizzato per associare la codifica dei targets in maniera progressiva.

La directory in cui devono essere presenti le immagini dei targets codificati deve contenere i files in formato PNG con un nome nel formato `sigma3Dtarget_xxx.png` in cui `xxx` indica la codifica del targets. Nella cartella deve essere presente anche un file di immagine sempre in formato PNG con il nome `sigma3Dtarget_Uncoded.png`. Questo file è l'immagine di un target non codificato che si presenta come in Figura X dove ognuno dei cerchi presenti ha le stesse dimensioni del cerchio presente al centro dei targets codificati.

L'immagine in Fig. 4 viene utilizzata come texture dei targets che non devono presentare codifica e nel caso in cui nella cartella non sia stato trovato il target corrispondente alla codifica cercata.

La funzione per la creazione dei targets è la seguente:

```
def createtargets(self):
        targetspath = bpy.context.scene.targetspath
        targetsize = bpy.context.scene.target_size
```

```
        m1shapefilename = bpy.context.scene.M1_shape
        targetpositionsfilename = bpy.context.scene.
target_positions

        if targetspath:
            srt_utility.Settings.targetpath =
targetspath

        normal_vector = srt_utility.Settings.
NORMAL_VECTOR
        M1_d = 1
        target_z_offset = -0.18
        targets = {}
        values = srt_utility.createtargetslist()

        curve_values = srt_utility.
extractshapecurvepoints(filename = m1shapefilename)
        curve_values_min = curve_values[::M1_d]
        #create targets
        for k in values:
            if normal_vector:
                targets["{:0>3d}".format(k[0])] = self
.createtarget(targetsize = targetsize, location = (
k[1:4]), rotationangle = (k[4:7]), z_offset =
target_z_offset)
            else:
                targets["{:0>3d}".format(k[0])] = self
.createtarget(targetsize = targetsize, location = (
k[1:4]), z_offset = target_z_offset)
        print("targets created")
        #applay targets textures
        for k in targets.keys():
            self.uvMapperTarget(targets[k], k)
            bpy.ops.mesh.select_all(action="DESELECT")
        print("targets texture created")

    def createtarget(self, location = (0.,0.,0.),
rotationangle = (0.,0.,0.), targetsize = .06,
thickness = 0.000007, z_offset = 0.0):
        #targetsize (central circle) unit - meter
        #thickness unit - meter
        sticksize = targetsize*4
        tmplocation = (location[0], location[1],
location[2]+z_offset)
        bpy.ops.mesh.primitive_cube_add(location =
tmplocation)
        target = bpy.context.active_object
        bpy.ops.transform.resize(value=(sticksize,
sticksize, thickness))
        target.rotation_euler = rotationangle
```

```
        return target
```

La funzione `createtargets()` aggiunge al dictionary `targets` i targets che vengono via via creati dalla funzione `createtarget()`. Dopo la loro creazione viene applicato ad ognuno di essi una texture corrispondente alla codifica assegnata. La funzione `srt_utility.createtargetslist()` della libreria `srt_utility` si occupa di creare per ogni target il numero identificativo della codifica, la terna corrispondente alla posizione e la terna corrispondente alla rotazione di eulero XYZ

```
def createtargetslist(uncoded = True):
    #create the target list composed by [target_code,
   x_position, y_position, z_position, euler_x,
   euler_y, euler_z]
    M1_d = 1
    curve_YZ_filename = Settings.M1_shape_YZ_filename
    actuator_positions_file = Settings.
   actuator_position_filename

    targets = []

    curve_values = []
    try:
        curve_values = extractshapecurvepoints(
   filename = curve_YZ_filename)
        if len(curve_values) > 0:
            curve_values_min = curve_values[::M1_d]
        else:
            print(curve_YZ_filename + " File empty")
    except:
        print(curve_YZ_filename + " File Error")
        return
    xy_points = []
    try:
        xy_points = extractactuatorpositionpoints(
   filename = actuator_positions_file)
        if len(xy_points) == 0:
            print(actuator_positions_file + " File
   empty")
    except:
        print(actuator_positions_file + " File Error")
        return

    targets_point, target_angle =
   returntargetpositionandangle(curve_values_min, rays
    = 32, circles = 21, xy_points=xy_points)
    #set the proportion between coded and uncoded
   target (1 - every target are coded)
    target_coded_module = 1
    index = 0
    index2 = 1000 #start uncoded identifier
```

```
    targetcode = 2 #start code
    for p in targets_point:
        if index % target_coded_module == 0:
            #create coded target
            targets.append([targetcode]+list(p) + list
(target_angle[index]))
            targetcode +=1
        else:
            #create uncoded target
            targets.append([index2]+list(p) + list(
target_angle[index]))
            index2 += 1
        index +=1
    return targets
```

L'identificativo sarà un numero maggiore di 1000 nel caso di targets non codificati. La funzione `returntargetpositionandangle()` calcola la posizione del singolo target sulla base delle coordinate presenti nel file con la posizione dei targets visto in precedenza.

```
def returntargetpositionandangle(arc, rays=8, circles
=8, xy_points=[]):
    #return in targetpoints and normalvectorsangles
    the position and the normal vector for each target

    #arc - the profile of parabole
    #rays - number of rays
    #circles - number of circles
    #xy_points - the list of coordinates XY of each
    targets. If it is empty calculate the coordinates
    by rays and circles

    targetpoints = []
    normalvectorsangles = []
    if len(xy_points) == 0:
        #xy_points not set

        #calculate the angle of each ray
        rayangle = [x * math.pi * 2 / rays for x in
range(rays)]
        index = 0
        #split each ray in circles segments
        segment = np.ceil(len(arc)/circles)

        for i in rayangle:
            #make the rotation matrix for i angle
            R = np.array([[np.cos(i), -np.sin(i), 0],
                          [np.sin(i), np.cos(i), 0],
                          [0,0,1]])
            index = 1
```

```
            for p in arc[1:-1]:
                if index % segment == 0:
                    #for each segment in each ray
place a target rotate by R
                    targetpoints.append(np.dot(R, p))
                    #calculate the normal vector for
the target
                    normal, vectorangle = returnnormal
(arc[index-1], arc[index+1], R)
                    #normalvectorsangles.append(
vectortoeuler(normal))
                    normalvectorsangles.append(
vectorangle)
                index+=1
    else:
        for p in xy_points:
            v1_u = [1,0]
            v2 = p[1:3]
            v2_u = unit_vector(v2)
            #calculate the distance between the point
p and the origin
            p_distance = np.linalg.norm(v2)
            #calculate the angle of normal vector of p
            p_angle = np.arccos(np.dot(v1_u, v2_u))
            if np.isnan(p_angle):
                if (v1_u == v2_u).all():
                    p_angle = 0.0
                else:
                    p_angle = np.pi
            elif v2[1] <= 0:
                p_angle += np.pi

            #make the roation matrix of angle p_angle
            R = np.array([[np.cos(p_angle), -np.sin(
p_angle), 0],
                          [np.sin(p_angle), np.cos(
p_angle), 0],
                          [0,0,1]])
            index = 0
            #take the index of the point of arc at the
 distance p_distance
            for arcpoint in arc[1:-1]:
                if arcpoint[1] >= p_distance:
                    break
                index += 1
            targetpoints.append(np.dot(R, arc[index]))
            #calculate the normal vector in index
point by previous e next point in arc rotated by R
            normal, vectorangle = returnnormal(arc[
index-1], arc[index+1], R)
```

```
            normalvectorsangles.append(vectorangle)
    return targetpoints, normalvectorsangles
```

La giusta inclinazione del target è stata calcolata prendendo in considerazione il seguente semplice algoritmo:

1. Calcolo della distanza d dall'origine sul piano XY del punto preso dalla lista delle posizioni dei targets.
2. Calcolo del punto sulla generatrice $g(i)$ corrispondente alla distanza d dal centro dell'origine.
3. Calcolo della normale $\vec{v}$ sul punto $g(i)$ utilizzando i punti $g(i-1)$ e $g(i+1)$.
4. Posizionamento del target con il centro nel punto $g(i)$ e con normale $\vec{v}$

La funzione per la creazione e l'applicazione delle texture è la seguente:

```
def uvMapperTarget(self, obj, targetcode,
    backgroundmaterialindex = None):

        targetname, materialname, texturename,
    target_path  = srt_utility.returntargetname(
    targetcode)

        if not materialname in bpy.data.materials:
            material = bpy.data.materials.new(
    materialname)
        else:
            material = bpy.data.materials.get(
    materialname)
        obj.name = targetname

        material.use_nodes = True
        nodes = material.node_tree.nodes
        links = material.node_tree.links
        node = nodes[1]

        image = bpy.data.images.load(target_path)
        textureNode = nodes.new(type="
    ShaderNodeTexImage")
        textureNode.image = image
        links.new(textureNode.outputs[0], node.inputs
    [0])
        obj.data.materials.append(material)

        bpy.context.scene.objects.active = obj

        bpy.ops.object.mode_set(mode = "OBJECT")
        try:
            bpy.ops.object.select_all(action="DESELECT
    ")
        except:
```

```
            bpy.ops.object.mode_set(mode = "EDIT")
            bpy.ops.object.select_all(action="DESELECT
")
            bpy.ops.object.mode_set(mode = "OBJECT")
            bpy.ops.object.select_all(action="DESELECT
")
        #    pass
        obj.data.polygons[2].select = True
        obj.select = True
        bpy.ops.object.mode_set(mode = "EDIT")
        bpy.ops.uv.unwrap()
        obj.select = False
        obj.data.polygons[2].select = False

        bpy.context.scene.objects.active = None
        return 0
```

La funzione crea una nuova texture per ogni singolo target codificato, più una texture riutilizzata per tutti i targets non codificati. Questo crea un numero elevato di texture che crea dei problemi nell'utilizzo del linguaggio CUDA[3] per il rendering delle scene. Questo problema verrà illustrato più avanti.

La texture viene applicata a tutte le facciate del target e ridimensionata adattarsi alla forma di ognuna delle facciate.

4.2 Posizionamento del modello 3D di M1

Per posizionare il modello 3D della superficie primaria M1 di SRT devono essere indicati:

- il file del modello in formato PLY
- lo spostamento lungo l'asse Z che verrà applicato al modello

Il modello 3D di SRT utilizzato dal simulatore è stato realizzato dalla SJM TECH 3D e comprende le parti del radiotelescopio che interessano lo specchio primario e gli oggetti che mettono in ombra la superficie primaria stessa e cioè lo specchio secondario, il quadrupode, la camera del gregoriano e l'APEX. Questo permette, durante la realizzazione dei rendering, di prendere in considerazione l'effetto delle parti di SRT che ostruiscono la visuale della superficie primaria da parte delle camere virtuali.

Il modello fornito utilizza, per lo specchio primario, un profilo che non è quello reale della generatrice, ma quello ideale di un paraboloide. Questo crea una differenza tra il profilo reale e quello ideale causando l'intersezione delle due superfici.

Il modello 3D viene allineato con l'origine della generatrice e traslato lungo l'asse Z di $-5,85$ metri. Questo valore è stato ottenuto come differenza tra il sistema di riferimento del modello 3D e il profilo della generatrice.

La funzione per il posizionamento del modello 3D della superficie primaria non necessita di commenti.

```
def place_m1_3d(self):
    z_offset = bpy.context.scene.M1_Z_offset
```

```
        filepath = bpy.context.scene.M1_3D_file
        if filepath:
            m1_path = filepath
        else:
            m1_path = srt_utility.returnM1filepath()
        bpy.ops.import_mesh.ply(filepath=m1_path)
        m1_surface = bpy.context.active_object
        #traslate m1_surface on Z
        m1_surface.location = (0.0, 0.0, z_offset)
```

4.3 Creazione delle camere

Per le camere virtuali nel simulatore deve essere indicato:

- il file di testo con la posizione e l'orientamento delle singole camere

Il file di testo con la posizione e l'orientamento delle singole camere ha il seguente formato:

```
2.659491177     -2.608852935    4.5
                16.34447739     -16.27336716
   5.72492225
0.00347775      -3.751388035    4.5
                -0.007220909    -23.06791019
   5.725184899
-2.638837388    -2.630483926    4.5
                -16.30935989    -16.31689539
   5.724862578
-3.784199519    -0.036310641    4.5
                -23.06703621    -0.015963116
   5.724862578
```

Il file non ha una riga di intestazione e le righe non hanno un'etichetta identificativa; ogni riga rappresenta una camera per la quale i primi tre valori indicano la posizione nello spazio espressa in metri. Gli altri tre valori possono rappresentare il vettore direzione o il vettore della rotazione di Eulero ZYX. Questa opzione è indicata dalla variabile `srt_utility.Settings.camera_direction_by_vector` che di default è importata a `True`. Di seguito la funzione per la creazione delle camere.

```
    def createcameras(self):
        cameraspath = bpy.context.scene.
    camera_positions

        camera_f = srt_utility.Settings.camera_f
        camera_f_stop = srt_utility.Settings.
    camera_f_stop
        sensor_width = srt_utility.Settings.
    sensor_width
        sensor_height = srt_utility.Settings.
    sensor_height
        factor = 0.001 # 1 = 1m
```

```
        camera_direction_by_vector = srt_utility.
Settings.camera_direction_by_vector
        camerapositionlist = srt_utility.
createcameralist(filename = cameraspath)

        cameralist = []
        index = 1
        firstcameradistance = 0
        tmp_factor = factor
        if camera_direction_by_vector:
            tmp_factor = 1
        for c in camerapositionlist:
            cameralocation = (([x*tmp_factor for x in
c[0:3]]))
            rotationcamera = Vector(c[3:])
            if index == 1:
                #take distance from world center
                firstcameradistance = srt_utility.
calculatedistance([x*tmp_factor for x in c[0:3]])
            cameralist.append(self.createcamera("
Camera{:0>3d}".format(index), location =
cameralocation, rotationangle = rotationcamera,
FocusDistance = firstcameradistance, f = camera_f,
f_stop=camera_f_stop, sensor_width = sensor_width,
sensor_height = sensor_height))
            if camera_direction_by_vector:
                self.look_at(cameralist[-1],
rotationcamera)
            index += 1

    def look_at(self, obj_camera, point):
        loc_camera = obj_camera.matrix_world.
to_translation()

        direction = point - loc_camera
        # point the cameras '-Z' and use its 'Y' as up
        rot_quat = direction.to_track_quat('-Z', 'Y')

        # assume we're using euler rotation
        obj_camera.rotation_euler = rot_quat.to_euler(
"ZYX")
```

Per la creazione delle camere devono essere definiti i seguenti parametri:

- `camera_f` - la lunghezza focale della camera espressa in millimetri
- `camera_f_stop` - l'apertura del diaframma espressa in F/stop
- `sensor_width` - la larghezza del sensore della camera espresso in millimetri

- `sensor_height` - l'altezza del sensore della camera espresso in millimetri

questi parametri vengono presi dalla classe `srt_utility.Settings` e devono corrispondere alle caratteristiche della camera fotogrammetrica che si vuole utilizzare nel simulatore.

Per dare più realismo alle prese sintetiche è stata definita una distanza di messa a fuoco fissa tramite la variabile `firstcameradistance` che viene impostata come la distanza della prima camera creata dall'origine del sistema di riferimento. Questa distanza di messa a fuoco verrà utilizzata per tutte la camere create successivamente.

Ad ogni camere viene associato un nome nel formato `CameraXXX` dove `XXX` indica il numero progressivo della camera creata seguendo l'ordine trovato nel file con la posizione e l'orientamento delle camere.

La funzione `look_at()` non fa altro che orientare la camere nella direzione del vettore corrispondente alla coordinate del punto `point` passato come argomento. La camera avrà come parametri di rotazione una rotazione di Eulero ZYX.

Di seguito la funzione per la creazione della singola camera:

```
 def createcamera(self, name, location = (0.,0.,0.)
, rotationangle = (0.,0.,0.), FocusDistance = 1000,
 f = 28.661, f_stop=22.0, sensor_width = 36.0,
sensor_height = 24.0):
    bpy.ops.object.camera_add(location = location)
    camera = bpy.context.active_object
    camera.name = name
    camera.rotation_mode = 'ZYX'
    camera.rotation_euler = rotationangle
    camera.data.dof_distance = FocusDistance
    camera.data.lens = f
    #camera.data.clip_start = 2.5
    camera.data.clip_end = float("inf")
    camera.data.cycles.aperture_type = "FSTOP"
    camera.data.cycles.aperture_fstop = f_stop
    camera.data.sensor_width = sensor_width
    camera.data.sensor_height = sensor_height
    return camera
```

Nella funzione oltre ad essere utilizzati i valori passati per le rispettive variabili viene anche impostato il parametro `camera.data.clip_end` che indica la distanza del piano di clipping lontano. Questo viene impostato su un valore infinito per rendere più realistica la creazione delle prese sintetiche.

4.4 Salvataggio su file delle informazioni sulla posizione dei targets

Per il salvataggio delle informazioni sui targets devono essere indicati:

- il nome del file di testo in cui vanno salvate le informazioni delle posizioni dei targets
- il numero di targets che verranno estratti per la calibrazione

- il nome del file di testo in cui vanno salvate le informazioni delle posizioni dei targets estratti per la calibrazione

Il file di testo che conterrà le posizioni dei targets avrà il seguente formato:

```
002      -585.9745144844055       4451.044082641602
           35.38946062326431
003      -2732.9936027526855      3561.7282390594482
           35.38946062326431
004      -4147.708415985107       1718.0440425872803
           35.38946062326431
005      -4451.041698455811       -585.9929919242859
           35.38946062326431
```

Il file non presenta una riga di intestazione. I quattro valori di ogni riga indicano il codice del target e le coordinate XYZ della posizione espresse in millimetri.

I calibration points sono il numero di targets che verranno estratti dalla lista precedente e che verranno utilizzati dal software AICON 3D Studio come punti di calibrazione.

La funzione è la seguente:

```
 def extracttargetpoints(self):
     calibration_filepath = bpy.context.scene.
target_calibratio_positions
     calibration_point_number = bpy.context.scene.
calibration_point_number
     position_filepath = bpy.context.scene.
target_code_positions
     lines = []
     codedtargets = []
     for o in bpy.data.objects:
         if re.match(r"sigma3D.arget.*", o.name):
             if re.match(r".*Uncoded.*", o.name):
                 newline = "1" + o.name.split("_")
[-1].replace("Uncoded.", "").replace("Uncoded", "
000") + "\t" + str(o.location[0]*1000) + "\t" + str
(o.location[1]*1000) + "\t" + str(o.location
[2]*1000)
             else:
                 newline = o.name.split("_")[-1] +
"\t" + str(o.location[0]*1000) + "\t" + str(o.
location[1]*1000) + "\t" + str(o.location[2]*1000)
                 codedtargets.append(newline)
             lines.append(newline)

     srt_utility.createfile(position_filepath,
lines)
     codedtarget_extracted = srt_utility.
extractrandomfromlist(codedtargets,
calibration_point_number)
```

```
        srt_utility.createfile(calibration_filepath,
    codedtarget_extracted)
```

Il file di testo che verrà creato avrà lo stesso formato del file visto prima. Da notare che al file va aggiunta la seguente riga di intestazione per fare in modo che sia utilizzabile dal software AICON 3D Studio.

```
[content order = NUM X Y Z DEC %46% DAT %9%]
```

4.5 Creazione delle prese sintetiche

Per la creazione delle prese sintetiche devono essere indicati:

- L'errore che verrà applicato alla posizione delle camere
- L'angolo di rotazione delle camere
- Il numero di rotazioni per asse e per direzione
- Il numero di files che verrà creato
- Il nome della directory in cui verranno salvati gli output dei rendering

Con i primi tre parametri vengono definiti: un errore random nell'intervallo $[-\epsilon, \epsilon]$, un angolo α, e un numero di spostamenti n. Ogni camera viene quindi ruotata lungo l'asse Z di un angolo α per n volte in senso orario rispetto alla sua posizione originale e per n volte in senso antiorario rispetto alla sua posizione originale. Per ogni spostamento viene aggiunto alla posizione della camera un errore random compreso tra $[-\epsilon, \epsilon]$ e viene generata una presa sintetica. Stessa cosa viene fatta lungo l'asse Y per un totale di $4n + 1$ prese sintetiche per ogni camera virtuale.

Nelle versioni successive dell'Addon, per rendere più realistico il posizionamento delle camere e il loro basculamento, verrà utilizzata per l'errore una distribuzione gaussiana.

Ogni presa viene salvata nella cartella di output. Il file ha un nome nella forma

`picture_CameraCCC_A_B.jpg`

dove `CCC` rappresenta il numero della camera, `A` lo step di spostamento lungo l'asse Y e `B` quello lungo l'asse Z. Lo step di spostamento è un indice compreso nell'intervallo $[0, 2n]$. Al termine della creazione di ogni file di rendering viene creato un file con lo stesso nome in formato testo (`picture_CameraCCC_A_B.jpg.txt`) dove sono indicate le coordinate della camera e gli angoli di Eulero ZYX della direzione della camera. Nel caso in cui nella cartella sia già presente un file con lo stesso nome questo non viene ricreato. La creazione delle prese si interrompe quando viene raggiunto il valore di "Files to render".

Al termine della creazione di tutte le prese viene generato un file di testo con le informazioni presenti in tutti i files di testo associati ad ogni presa.

In Fig. 5 si può vedere una presa sintentica output dell'addon.

Di seguito il codice della funzione di generazione delle prese sintetiche:

```
  def render_bascule(self):
```

```
    camera_location_error =  bpy.context.scene.
camera_location_error
    camera_rotation_angle =  bpy.context.scene.
camera_rotation_angle  #angle in degree
    camera_rotation_positions =  bpy.context.scene
.camera_rotation_positions
    render_path = bpy.context.scene.render_path
    number_of_rendering = bpy.context.scene.
render_file_numbers
    render_resolution_x = srt_utility.Settings.
render_resolution_x
    render_resolution_y = srt_utility.Settings.
render_resolution_y
    render_resolution_pencentage = srt_utility.
Settings.render_resolution_pencentage

    #set rendering parameters
    bpy.data.scenes["Scene"].render.resolution_x
= render_resolution_x
    bpy.data.scenes["Scene"].render.resolution_y
= render_resolution_y
    bpy.data.scenes["Scene"].render.
resolution_percentage  =
render_resolution_pencentage

    if not number_of_rendering:
        number_of_rendering = 0
    if camera_location_error:
        srt_utility.Settings.cameralocationerror =
 camera_location_error
    if camera_rotation_angle:
        srt_utility.Settings.camera_angle_bascule
= math.radians(camera_rotation_angle) #angle in
radians
    if camera_rotation_positions:
        srt_utility.Settings.
camera_rotation_positions =
camera_rotation_positions

    rendering_found = 0
    rendering_maked = 0

    for c in bpy.data.objects:
        angle = srt_utility.Settings.
camera_angle_bascule
        camera_rotation_positions = srt_utility.
Settings.camera_rotation_positions
        rot = []
        index = -camera_rotation_positions
        lines = []
```

```
        for i in range(2*camera_rotation_positions
+1):
            rot.append(angle*index)
            index +=1
        if c.type == "CAMERA":
            initial_location = c.location
            c.rotation_mode = "QUATERNION"
            c.rotation_mode = "XYZ"
            rotation_euler_initial_X = c.
rotation_euler[0]
            c.rotation_mode = "QUATERNION"
            c.rotation_mode = "YZX"
            rotation_euler_initial_Y = c.
rotation_euler[1]
            c.rotation_mode = "QUATERNION"
            c.rotation_mode = "ZYX"
            bpy.context.scene.objects.active = c
            area = None
            for ar in bpy.context.screen.areas:
                if ar.type == 'VIEW_3D':
                    area = ar
                    break

            new_ctx = bpy.context.copy()
            new_ctx["area"] = area
            index_Y = -camera_rotation_positions
            for a in [0,2]:
                if a == 0:
                    index_X = -
camera_rotation_positions
                    index_Y = 0
                else:
                    index_Y = -
camera_rotation_positions
                    index_X = 0
                for i in rot:
                    c.rotation_mode = "QUATERNION"
                    if a == 0:
                        c.rotation_mode = "XYZ"
                        c.rotation_euler[0] =
rotation_euler_initial_X + i
                    else:
                        c.rotation_mode = "YZX"
                        c.rotation_euler[1] =
rotation_euler_initial_Y + i
                    c.rotation_mode = "QUATERNION"
                    c.rotation_mode = "ZYX"
                    srt_utility.addlocationerror(c
, srt_utility.Settings.cameralocationerror)
```

```
                    filename = render_path + "
picture_{}_{}_{}.jpg".format(c.name, str(index_X+2)
, str(index_Y+2))
                    if number_of_rendering > 0:
                        if number_of_rendering ==
rendering_maked:
                            pass
                        else:

                            if not os.path.isfile(
filename):
                                bpy.ops.view3d.
object_as_camera(new_ctx)

                                bpy.data.scenes["
Scene"].render.filepath = filename
                                #print(bpy.data.
scenes["Scene"].render.filepath)

                                try:
                                    #bpy.ops.
object.mode_set(mode = "OBJECT")
                                    bpy.ops.render
.render(write_still=True)
                                    srt_utility.
saverenderfilepositions(c, filename)
                                    lines.append("
{}\t{}\t{}\t{}\t{}\t{}".format(c.location[0],

                                    c.location[1],

                                    c.location[2],

                                    c.rotation_euler[0],

                                    c.rotation_euler[1],

                                    c.rotation_euler[2])
)

rendering_maked += 1

                                except:
                                    print("
Rendering error")
                            else:
                                rendering_found +=
 1
                    c.location = initial_location
                    if a == 0:
```

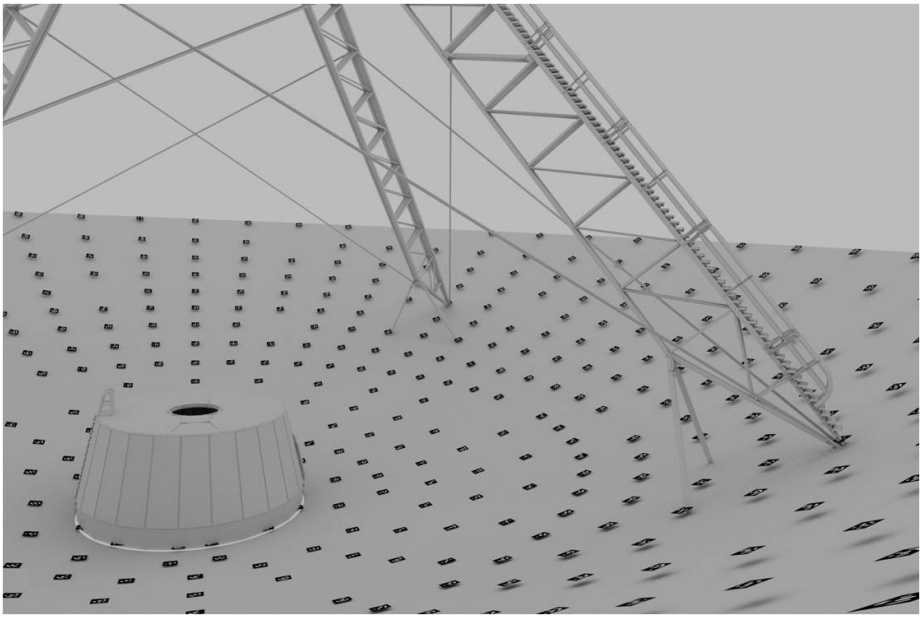

Figura 5: Esempio di presa sintetica realizzata tramite il simulatore

```
                        index_X += 1
                    else:
                        index_Y += 1
                c.rotation_mode = "QUATERNION"
                if a == 0:
                    c.rotation_mode = "XYZ"
                    c.rotation_euler[0] =
rotation_euler_initial_X
                else:
                    c.rotation_mode = "YZX"
                    c.rotation_euler[1] =
rotation_euler_initial_Y
                c.rotation_mode = "QUATERNION"
                c.rotation_mode = "ZYX"
                #set c in initial location
                c.location = initial_location
    f = srt_utility.openfile(filename + "FINAL.txt
", write=True)

    for line in lines:
        f.write(line + "\n")

    f.close()
```

4.6 Rimozione dalla scena degli oggetti presenti

Questa sezione mostra come avviene l'eliminazione dalla scena di tutte le camere presenti o tutte le meshs presenti. Le meshs comprendono i targets e il modello 3D di SRT.

```
 def delete_old_stuff(self):
     #remove from the scene all objects, meshs,
curves and cameras
     #remove materials and texture also
     remove_cameras = bpy.context.scene.
remove_cameras
     remove_meshs = bpy.context.scene.remove_meshs
     # escape edit mode
     if bpy.ops.object.mode_set.poll():
         bpy.ops.object.mode_set(mode='OBJECT')
     if remove_meshs:
         try:
             # delete all mesh objects
             bpy.ops.object.select_by_type(type='
MESH')
             bpy.ops.object.delete()
         except:
             print("Error delete mesh")

         try:
             # delete all poliline objects
             bpy.ops.object.select_by_type(type='
CURVE')
             bpy.ops.object.delete()
         except:
             print("Error delete curve")

         # delete all materials
         for i in bpy.data.materials.values():
             try:
                 bpy.data.materials.remove(i)
             except:
                 pass

         # delete all textures
         for i in bpy.data.textures.values():
             bpy.data.textures.remove(i)

         # delete all images
         for i in bpy.data.images.values():
             # delete image path, this is only
possible without a user
             i.user_clear()
             # delete all, except Render Result
             if i.name != "Render Result":
```

```
                bpy.data.images.remove(i)


    if remove_cameras:
        try:
            # delete all camera objects
            bpy.ops.object.select_by_type(type='
CAMERA')
            bpy.ops.object.delete()
        except:
            print("Error delete camera")
```

4.7 Considerazione sul motore di rendering

Il motore di rendering utilizzato è Cycles, incorporato all'interno di Blender, e tra i parametri impostati per il rendering ci sono la risoluzione del fotogramma e il formato di output.

Poiché Cycles è un motore di rendering unbiased e cioè che non introduce errore nel calcolo della luce in ogni punto dell'immagine, genera un calcolo potenzialmente infinito, e per limitarlo va impostato il parametro "samples". Questo parametro, se impostato su valori bassi, causa una presenza di rumore sull'immagine finale.

Tutti questi parametri sono memorizzata nella classe `srt_utility.Settings`

5 Risultati della simulazione

Per la validazione del simulatore è stato definito un ambiente in cui fossero presenti i targets codificati sulla posizione degli attuatori della superficie primaria, 8 camere sul bordo più esterno di M1, 8 camere sul bordo della camera del gregoriano e quattro camere in corrispondenza delle gambe del tetrapode. Con questa configurazione, e imponendo il basculamento, sono state generate le prese fotogrammetriche sintetiche che sono state utilizzate in AICON 3D Studio.

Il software è stato in grado di riconoscere i targets codificati e ha effettuato correttamente il bundle delle prese.

E' stato quindi preso l'output del software relativo alla posizione calcolata dei targets individuati e questo è stato confrontato con la posizione dei targets inseriti nel modello 3D. Il confronto delle posizioni punto per punto ha dato il seguente risultato che si traduce in RMS di 0,5345 mm

Come si può notare dal grafico in Fig. 6 la distribuzione dell'errore aumenta allontanandosi dal centro dello specchio primario, ed essendo la parte centrale quella che da maggiore contributo durante le osservazioni ad alte frequenze si possono considerare i risultati molto incoraggianti e nella direzione del progetto.

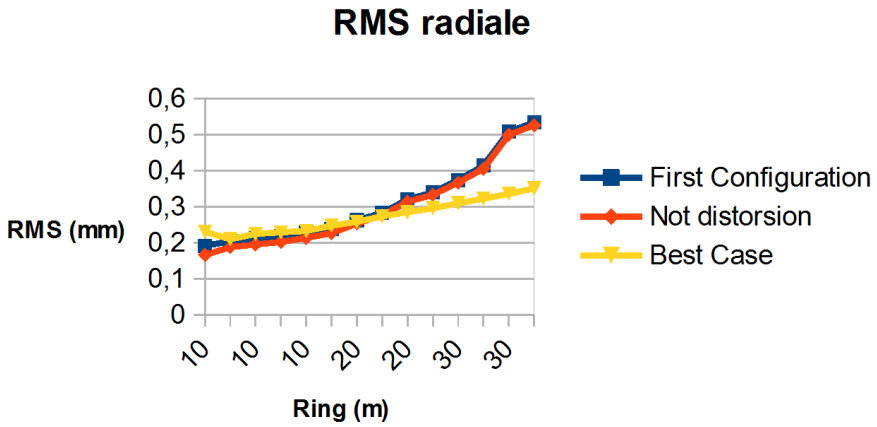

Figura 6: Distribuzione radiale dell'errore

6 Acknowledgment

Questo lavoro è stato condotto nell'ambito del progetto "Stima delle deformazioni del Sardinia Radio Telescope" CRP-26658 (L.R. 7/2007 Regione Autonoma della Sardegna)

7 Appendice

Riferimenti bibliografici

[1] https://www.blender.org/

[2] http://en.wikipedia.org/wiki/PLY

[3] http://www.nvidia.com/object/cuda_home_new.html

[4] http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=36631

[5] http://aicon3d.com/start.html?gclid=CJH_qo2h1cUCFQXLtAodgi0AuQ