UNIVERSITÀ DI PISA

Facoltà di Ingegneria

Laurea Specialistica in Ingegneria dell'Automazione

Tesi di laurea

Addressing pose estimation issues for application of machine vision to UAV Autonomous Aerial Refueling

Tesi di laurea svolta presso Department of Mechanical and Aerospace Engineering
West Virginia University (Morgantown)

Candidato:

Marco Mammarella ____________________

Relatori:

Prof. Mario Innocenti ____________________

Prof. Andrea Caiti ____________________

Sessione di Laurea del 19/07/2005
Archivio tesi Laurea Specialistica in Ingegneria dell'Automazione
Anno accademico 2004/2005
Consultazione consentita

Questo documento è il frutto di un lungo periodo di studio e di vita su cui mi sento in dovere di ringraziare:

- *Prof. Marcello Napolitano e Dr. Giampiero Campa per il supporto tecnico e morale fornito durante la realizzazione di questo progetto.*
- *I miei familiari per l'immenso supporto morale e la disponibilità finanziaria fornita.*
- *Sofia, Matteo, Tore e Gabriele per essermi stati vicini nelle innumerevoli avventure di questi anni ma anche nei momenti difficili.*
- *Tutti coloro che hanno contribuito a rendere questo periodo indimenticabile.*

ABSTRACT

The purpose of this thesis is to describe the results of an effort on the analysis of the performance of specific algorithms for the 'pose estimation' problem within the context of applying Machine Vision-based control laws for the problem of Autonomous Aerial Refueling (AAR) for UAVs. It is assumed that the MV-based AAR approach features several optical markers installed on specific points of the refueling tanker. However, the approach can be applied without any loss of generality to the more general case of the use feature extraction methods to detect specific points and corners of the tanker in lieu of the optical markers. The document proposes a robust 'detection and labeling algorithm' for the correct identification of the optical markers, which is then provided to the 'pose estimation' algorithm. Furthermore, a detailed study of the performance of two specific 'pose estimation' algorithms (the GLSDC and the LHM algorithms) is performed with special emphasis on required computational effort, robustness, and error propagation. Extensive simulation studies demonstrate the potential of the LHM algorithm and also highlight the importance of the robustness of the 'detection and labeling' algorithm. The simulation effort is performed with a detailed modeling of the AAR maneuver using the USAF refueling method.

SOMMARIO

Lo scopo di questa tesi è descrivere i risultati di uno studio riguardante l'analisi delle prestazioni di specifici algoritmi per il problema della stima della posizione in un contesto applicato ad una legge di controllo basata su Machine Vision (MV) per il problema del rifornimento aereo in modo autonomo (AAR) per veicoli aerei non pilotati (UAVs). Si assume che l'avvicinamento durante il rifornimento avvenga grazie a diversi markers ottici installati in punti specifici dell'aeromobile che fornisce il carburante (Tanker). Il metodo può comunque essere usato senza perdita di generalità in un caso dove si usa il metodo della feature extraction per trovare dei punti specifici sul contorno del Tanker al posto dei markers ottici. Il documento propone un algoritmo robusto per la corretta identificazione dei markers ottici, i quali vengono poi forniti agli algoritmi di stima della posizione. Inoltre si propone uno studio dettagliato di due specifici algoritmi per la stima della posizione (il GLSDC e LHM) dove si da particolare importanza al peso computazione, la robustezza e la propagazione dell'errore. Numerose simulazioni dimostrano il potenziale dell'algoritmo LHM evidenziando l'importanza della robustezza e dell'algoritmo di identificazione dei markers ottici. Le simulazioni sono state eseguite con un modello dettagliato della manovra di AAR usando il metodo proposto da USAF.

INDEX

FIGURE INDEX

TABLE INDEX

1. THE AUTONOMOUS AERIAL REFUELING PROBLEM

The strategic and tactical importance of Unmanned Aerial Vehicles (UAVs) for civil and military purposes has grown in recent years. The deployment of UAVs has been tested in current and recent overseas conflicts. Reducing costs and risks of human casualties is one immediate advantage of UAVs. An additional advantage is the possibility of avoiding troop deployment in enemy territory for dangerous rescue missions as is done currently with manned missions. It is envisioned that formations of UAV will perform not only intelligence and reconnaissance missions but provide close air support, precision strike, and suppression of enemy air defenses.

One of the biggest current limitations of deployed military UAVs is their limited aircraft range. In fact, today UAVs are not capable of overseas flight and need to be flown by ground troops deployed at limited distances from a combat scenario. Furthermore, terrain and weather factors can also determine how close to the targets the UAVs can be launched. Therefore, the acquisition of AAR capabilities for UAVs is a critical goal. To achieve these capabilities specific technical challenges need to be overcome.

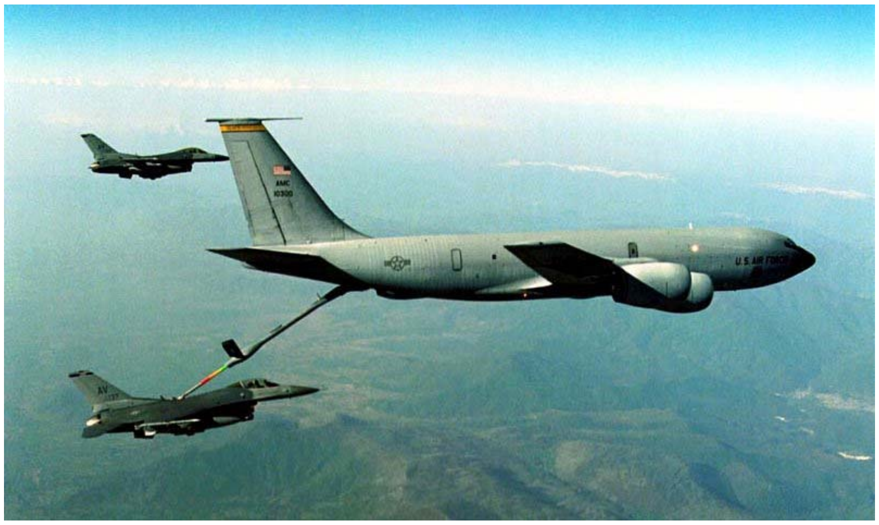


Fig. 1.1: aerial refueling using a boom system

Currently, there are two types of hardware set-ups used for aerial refueling. The first method is used by the US Air Force and is based on a refueling boom (Fig. 1.1); the second method is used by the US Navy as well as the armed forces of other NATO nations and is based on a "probe and drogue" setup, consisting of a refueling flexible hose with a flexible basket at the end (Fig. 1.2).

In recent years, the AAR problem has attracted the attention of many researchers [1][2][27]. In this effort a key issue is represented by the need of a high accuracy measurement of the relative Tanker-UAV relative distance and attitude in the final phase of docking and during the refueling. The use of MV technology has been proposed in addition or as an alternative to more conventional GPS technology. Particularly, a MV-based system has been proposed for close proximity operations of aerospace vehicles [37] and for the navigation of UAVs [38]. For the "probe and drogue" refueling system a MV-based system has been proposed in Refs. [1][2][27]. Within these studies, a fixed or variable number of visible optical markers is assumed to be available. On the other hand, temporary loss of visibility might occur due to hardware failures and/or physical interference between the UAV on-board camera and the markers due to the refueling boom itself and/or different structural components of the Tanker or just simply because the markers exit the visual range of the on-board camera.

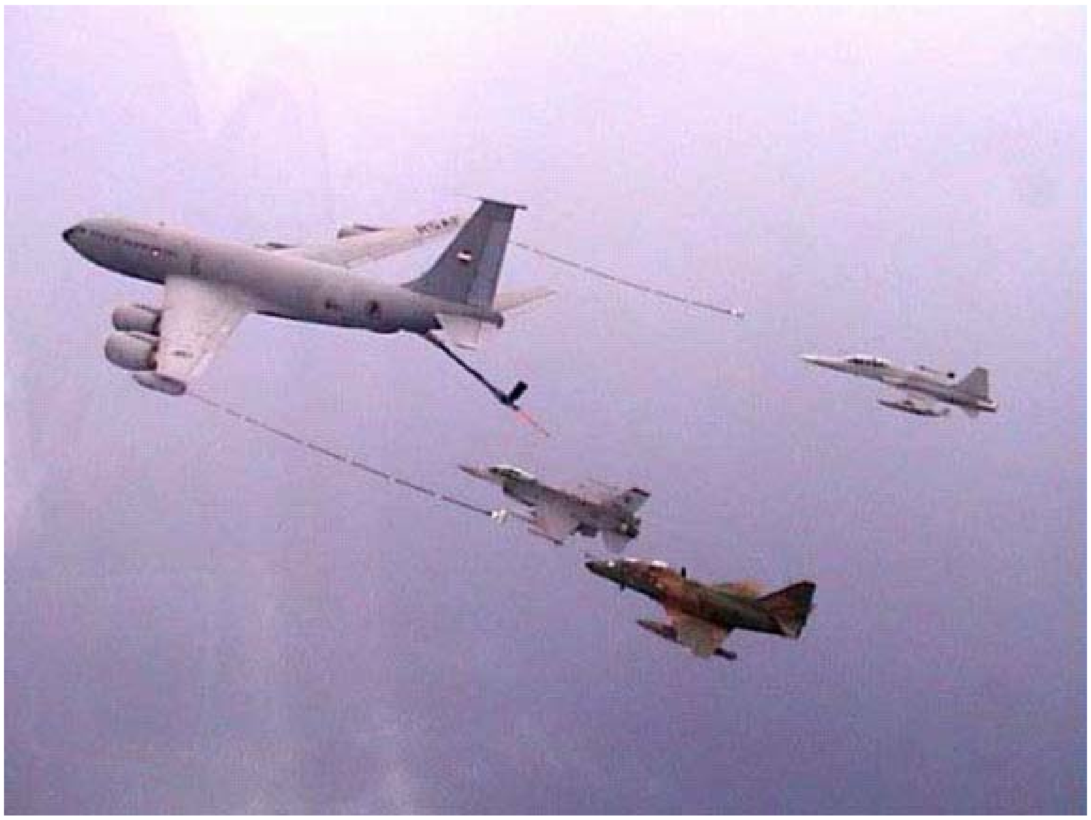


Fig. 1.2: aerial refueling using a probe and drogue system

In this effort the AAR problem is addressed for the “refueling boom” method. In this case, the objective is to guide the UAV to a defined 3-D Window (3DW) below the Tanker where the boom operator can then manually proceed to the docking of the refueling boom with the UAV fuel receptacle followed by the actual refueling phase (Fig.1.3).

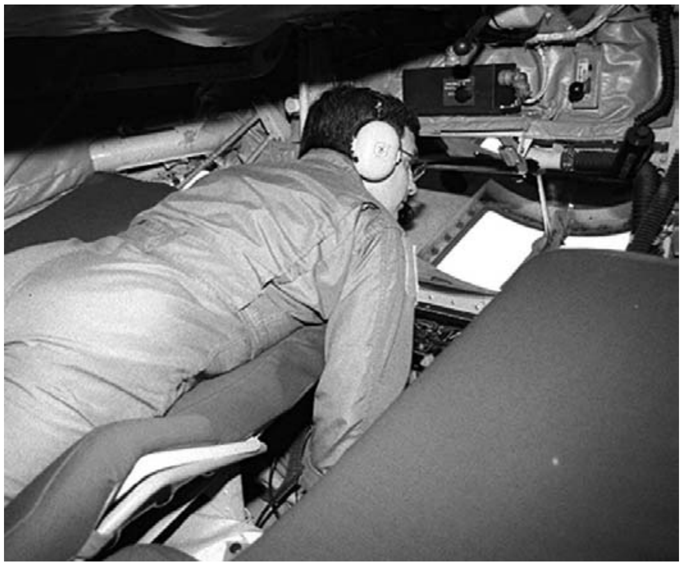


Fig. 1.3: boom operator on Boeing KC-135R

A specific docking control scheme featuring a fusion of GPS and MV distance measurements is proposed in this thesis. The MV estimation algorithms studied are capable of handling temporary loss of visibility of the markers. A detailed simulation environment has been designed providing accurate modeling for the drogue flexibility, the wake effects from the Tanker on the UAV, the atmospheric turbulence, the UAV trajectory constraints, as well as the GPS and MV measurements errors. Extensive simulations of the various aspects of the proposed docking scheme are analyzed and discussed; the UAV has been characterized with the parameters of the ICE-101 aircraft [39][38], the Tanker by a Boeing KC-135R aircraft [37].

A simulation environment for the AAR problem (Fig.1.4) is developed and analyzed, the simulation uses Virtual Reality Toolbox® (VRT) of Matlab for the image representation, a simulated camera is included in the simulation, it capture the VRT image and process the image to extract the position of the points, the MV system uses this data to provide the relative distance between camera and Tanker. The analysis of the MV system is the main purpose of this effort, using a particular emphasis for the pose estimation algorithms: Gaussian Least Squares Differential Correction (GLSDC) and the Lu, Hager and Mjolsness algorithm (LHM). The differences between the two algorithms will be analyzed, highlighting and the most important results.

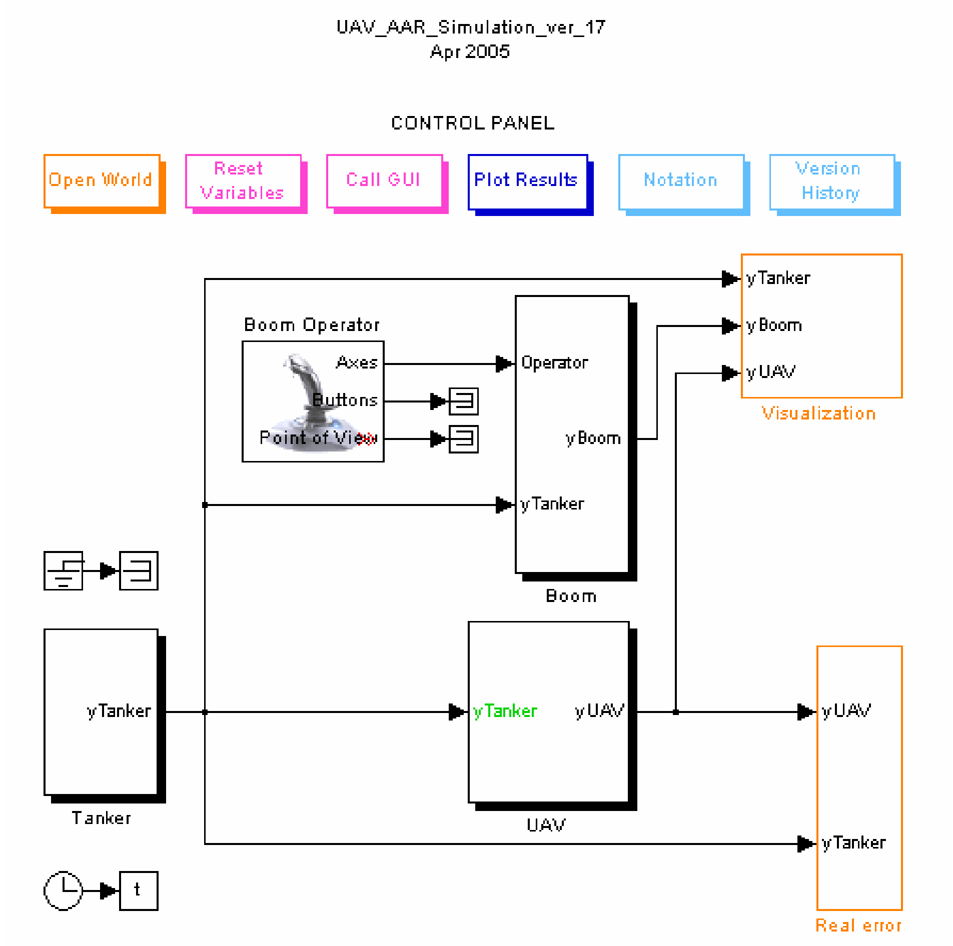


Fig. 1.4: AAR simulation

1.1 Reference frame

The relevant reference frames, the problem formulation, sensors and distance vectors will be described in this section.

The general scheme of the (Tanker + refueling boom + UAV) system is shown in Fig. 1.5

The study of the AAR problem requires the definition of specific Reference Frames (RFs); where ERF is the earth fixed reference frame; TRF and URF are the body fixed RF for the Tanker and the UAV respectively; these RFs are applied in the center of mass of the aircrafts. Note, CRF refers to a fixed UAV camera RF. To make the docking problem invariant with respect to the nominal heading of the aircraft, an additional fixed frame MRF is defined; this frame is rotated of the nominal heading angle ψ_0 with respect to the ERF. In this thesis, the following notation will be used: ${}^{E}R$ represents a point R expressed within ERF, ${}^{E}AB$ represents the vector from point A to point B expressed within ERF, while the matrix ${}^{E}T_{T}$ represents the homogeneous transformation matrix that transform a vector/point expressed within TRF in a vector/point expressed within ERF.

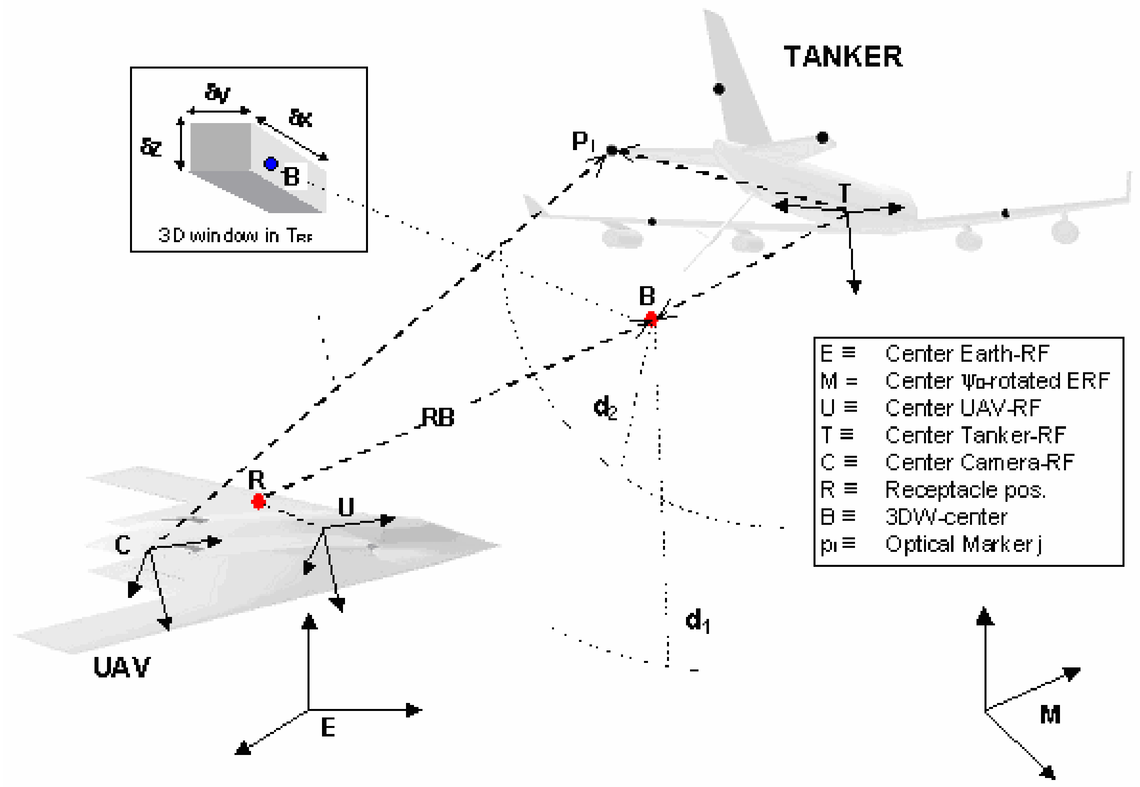


Fig. 1.5: reference frames in AAR problem

1.2 Problem formulation

The objective is to guide the UAV such that its fuel receptacle (point R in Fig.1.5) tracks the center of a 3-dimensional window (3DW) under the tanker (point B). Once the UAV fuel receptacle reaches and remains within this 3DW, the boom operator is assumed to take control of the refueling operations. It should be underlined that point B is fixed in the TRF with the dimensions of the 3DW ($\delta x, \delta y, \delta z$) presented in Tab 1.1 being an important design parameter. It is assumed that the tanker and the UAV share a data communication link. The UAV is equipped with a digital camera along with MV algorithms acquiring the images of the tanker and the point-wise markers (points P_j) installed on the tanker itself.

	Desired (meter)	**Limit (meter)**
δx	±0.40	±2.10
δy	±1.87	±2.10
δz	±0.90	±2.56

Tab. 1.1: 3DW dimension specification

Actually the desired parameters δx, δy and δz are respected both from the GLSDC and LHM algorithms, the result are presented in Fig 1.6 -1.8 and in Tab. 1.2 where a mean and a standard deviation of the 3DW distance are shown for the two algorithms with a initial position of the UAV aircraft x_0=[-50, -20, 30] within TRF.

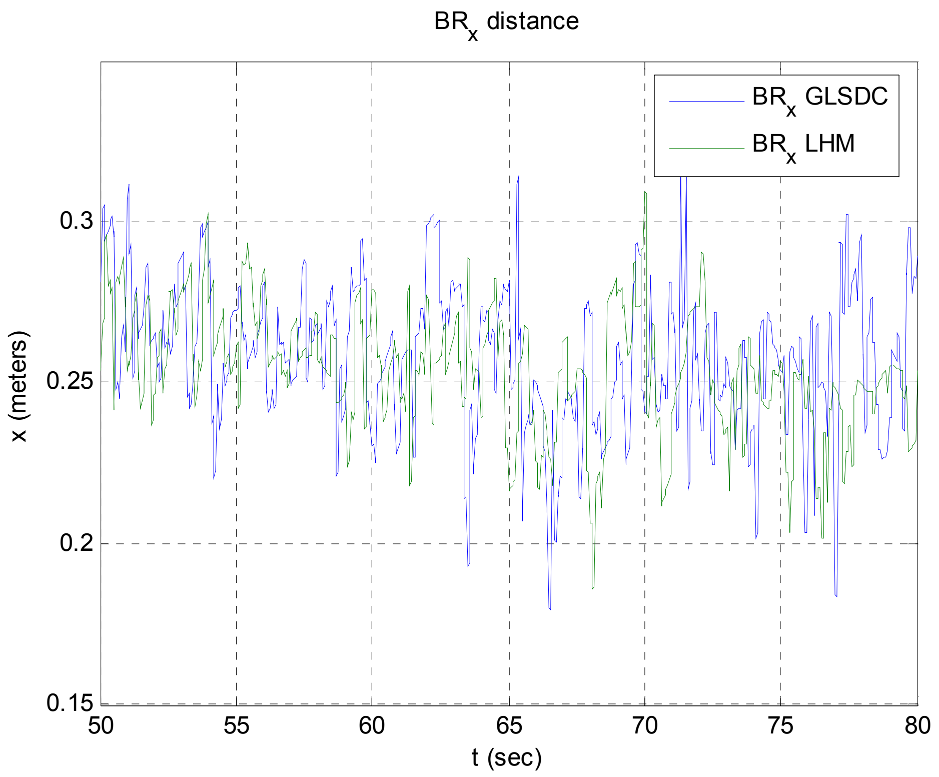


Fig. 1.6: coordinate x of BR distance in MRF

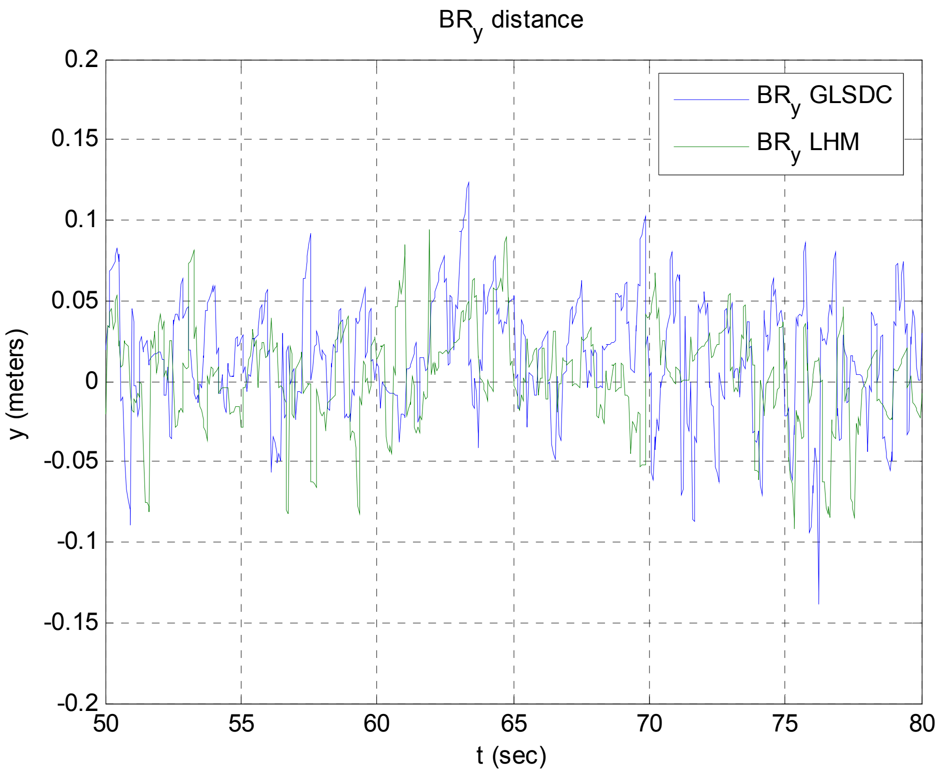


Fig. 1.7: coordinate y of BR distance in MRF

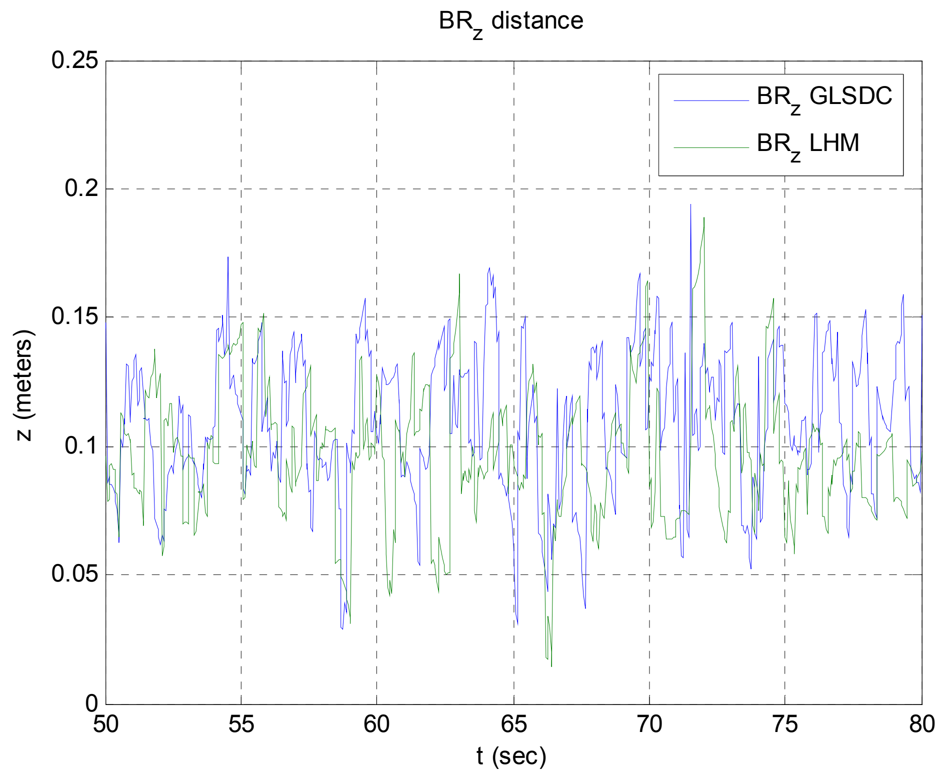


Fig. 1.8: coordinate z of BR distance in MRF

	Mean value (η)		Standard Deviation (σ)	
	GLSDC (meter)	**LHM (meter)**	**GLSDC (meter)**	**LHM (meter)**
δx	0.2581	0.2539	0.2160	0.2166
δy	0.0153	0.0034	0.0943	0.0898
δz	0.1100	0.0966	0.0949	0.0875

Tab. 1.2: 3DW mean distance and std with GLSDC and LHM algorithms

An evaluation of the tracking error is provided for the GLSDC and LHM algorithms in Fig.1.9 - 1.10 and in Tab.1.3, where we consider the tracking error as the difference between the actual position of the aircraft and the nearest point in the whole trajectory. The trajectory is considered to be simply a set of points, the time dependency is not considered for the tracking error calculation. An analysis of the tracking error shows that the aircraft follows the trajectory without major differences between the two algorithms.

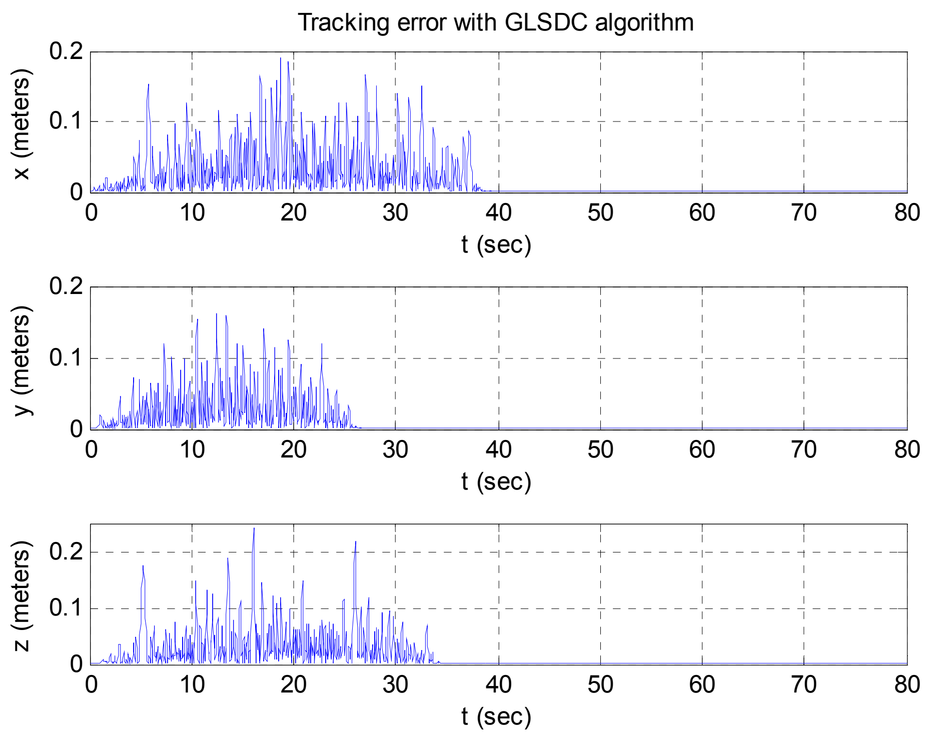


Fig. 1.9: tracking error with GLSDC algorithm

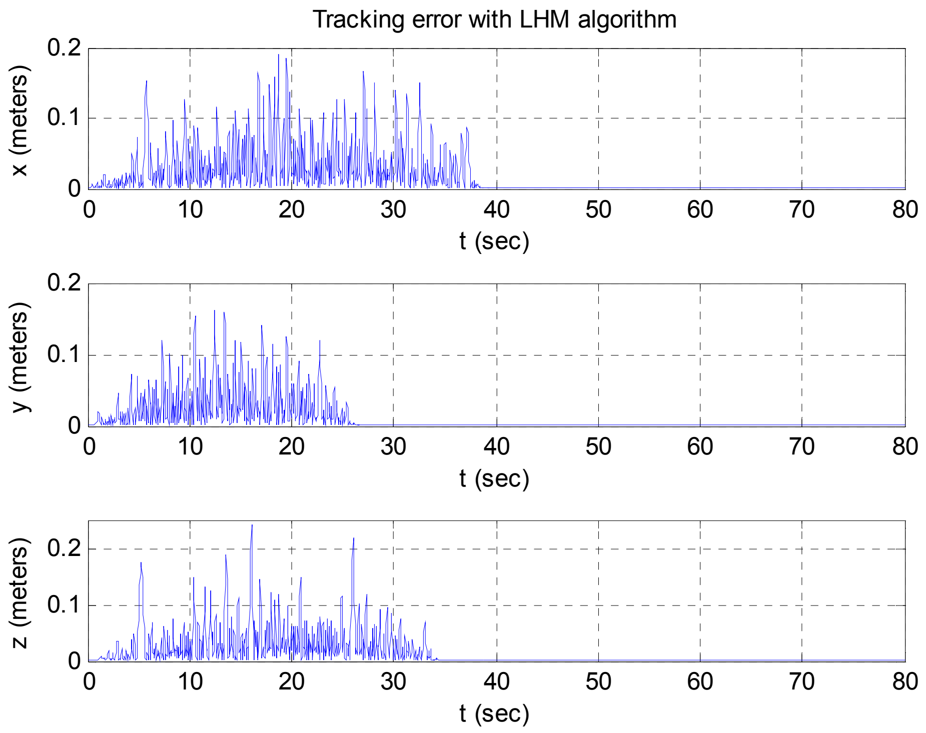


Fig. 1.10: tracking error with LHM algorithm

	GLSDC (meter)	**LHM (meter)**
x	$1.3831*10^{-2}$	$1.4180*10^{-2}$
y	$1.0154*10^{-2}$	$0.9996*10^{-2}$
z	$1.2835*10^{-2}$	$1.3674*10^{-2}$

Tab. 1.3: mean of tracking error

2. THE TANKER

The Tanker used in the AAR simulation is the Boeing KC-135R Stratotanker, this is among the most common aircraft used for this purpose.

The KC-135 Stratotanker's primary mission is to refuel long-range aircraft. It also provides aerial refueling support to Air Force, Navy, Marine Corps and allied aircraft. Four turbojets, mounted under wings swept 35 degrees, power the KC-135. Nearly all internal fuel can be pumped through the tanker's flying boom, the KC-135's primary fuel transfer method. A special shuttlecock-shaped drogue, attached to and trailed behind the flying boom, is used to refuel aircraft fitted with probes. An operator stationed in the rear of the plane controls the boom. A cargo deck above the refueling system holds passengers or cargo. Depending on fuel storage configuration, the KC-135 can carry up to 37,350 kilograms of cargo.

The KC-135 tanker fleet made an invaluable contribution to the success of Operation Desert Storm in the Persian Gulf, flying around-the-clock missions to maintain operability of allied planes. The KC-135s form the backbone of the Air Force tanker fleet, meeting the aerial refueling requirements of bomber, fighter, cargo and reconnaissance forces, as well as the needs of the Navy, Marines and allied nations.

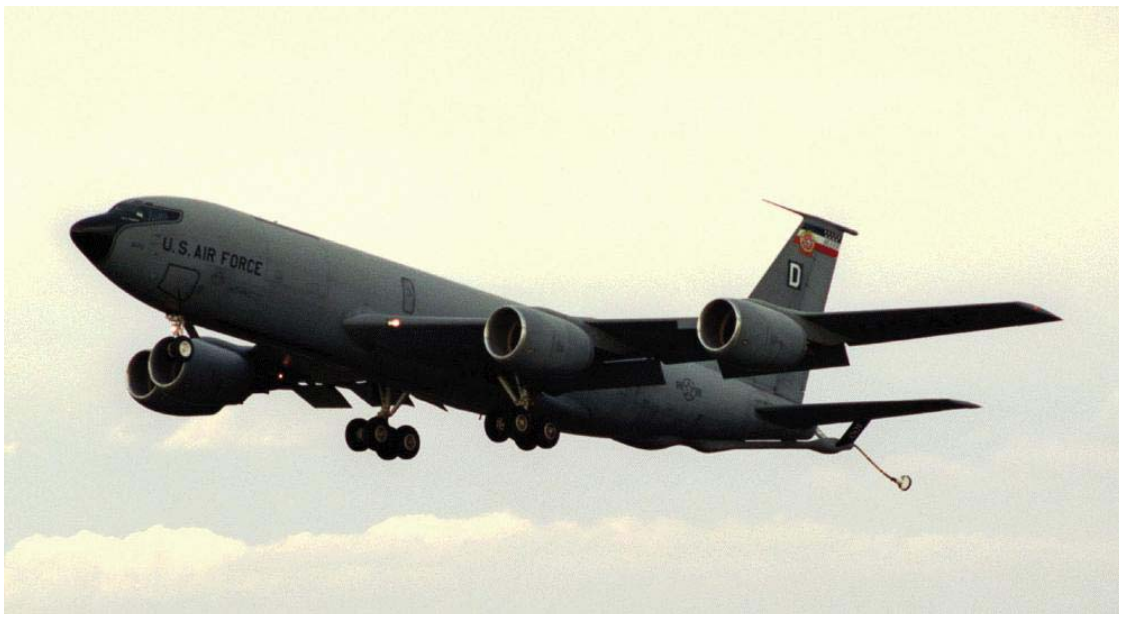


Fig. 2.1: Boeing KC-135R front view

Because the KC-135A's original engines are of 1950s technology, they don't meet modern standards of increased fuel efficiency, reduced pollution and reduced noise levels. By installing new, CFM56 engines, performance is enhanced and fuel off-load capability is dramatically improved. In fact, the modification is so successful that two-re-engined KC-135Rs can do the work of three KC-135As. This improvement is a result of the KC-135R's lower fuel consumption and increased performance which allow the tanker to take off with more fuel and carry it farther. Since the airplane can carry more fuel and burn less of it during a mission, it's possible to transfer a much greater amount to receiver aircraft.

The quieter, more fuel-efficient CFM56 engines are manufactured by CFM International, a company jointly owned by SNECMA of France, and General Electric of the U.S. The engine is an advanced-technology, high-bypass turbofan; the military designation is F108-CF-100. Related system improvements are incorporated to improve the modified airplane's ability to carry out its mission, while decreasing overall maintenance and operation costs. The modified airplane is designated a KC-135R.

Because the KC-135R uses as much as 27 percent less fuel than the KC-135A, the USAF can expect huge fuel savings by re-engining its fleet of KC-135s - about $1.7 billion over 15 years of operation. That's enough to fill the gas tanks of some 7.7 million American cars each year for a decade and a half. Annual savings are estimated to be about 2.3 to 3.2 million barrels of fuel, about three to four percent of the USAF's annual fuel use. This equals the fuel needed to provide electrical power for 145 days to a city of 350,000 to 400,000.

Re-engining with the CFM56 engines also results in significant noise reductions. Area surrounding airports exposed to decibel noise levels is reduced from over 240 square miles to about three square miles. This results in a reduction in the noise impacted area of more than 98 percent. Maximum take-off decibel levels drop from 126 to 99 decibels. This meets the tough U.S. Federal Air Regulation standards -- a goal for commercial aircraft operated within the U.S. In addition, smoke and other emission pollutants are reduced dramatically.

Boeing has delivered approximately 400 re-engined KC-135Rs and is under contract for about 432 re-engine kits. Each kit includes struts, nacelles, 12.2 miles of wiring, and other system modification components. Engines are purchased directly by the Air Force from CFM International.

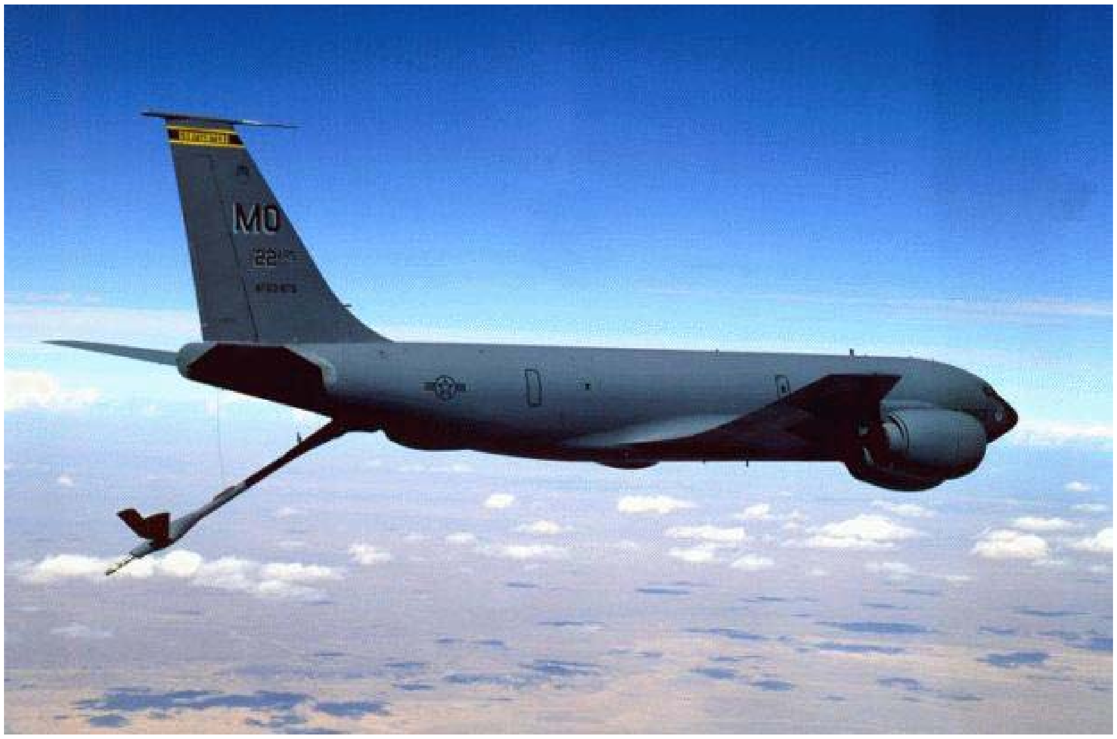


Fig. 2.2: Boeing KC-135R rear view

Boeing has completed work on a program to re-engine all KC-135As in the Air Force Reserve and Air National Guard fleet - a total of 161 airplanes. In that modification program, which began in 1981, KC-135As were modified with refurbished JT3D engines taken from used, commercial 707 airliners. After modification, the airplanes are designated KC-135Es. This upgrade, like the KC-135R program, boosts performance while decreasing noise and smoke pollution levels. The modified KC-135E provides 30 percent more powerful engines with a noise reduction of 85 percent.

The program included acquisition of used 707s, procurement of purchased parts and equipment, basic engineering, some parts manufacturing, and refurbishment and installation of the engines, struts and cowling. Kits also included improved brakes, cockpit controls and instruments.

2.1 Aircraft model

The used model for the Tanker has the dynamic characteristic of the Boeing KC-135R, the aircraft has to assume for the refueling a steady state equivalent to a rectilinear trajectory, a constant Mach number of 0.65 and an altitude (H) of 6,000 m. This allows simplifying the Tanker dynamics as described in section 3.1. The lateral-directional motion is eliminated by the dynamics inasmuch the aircraft has just a longitudinal motion. The longitudinal motion has a stable dynamics and the Tanker does not need an internal stability control.

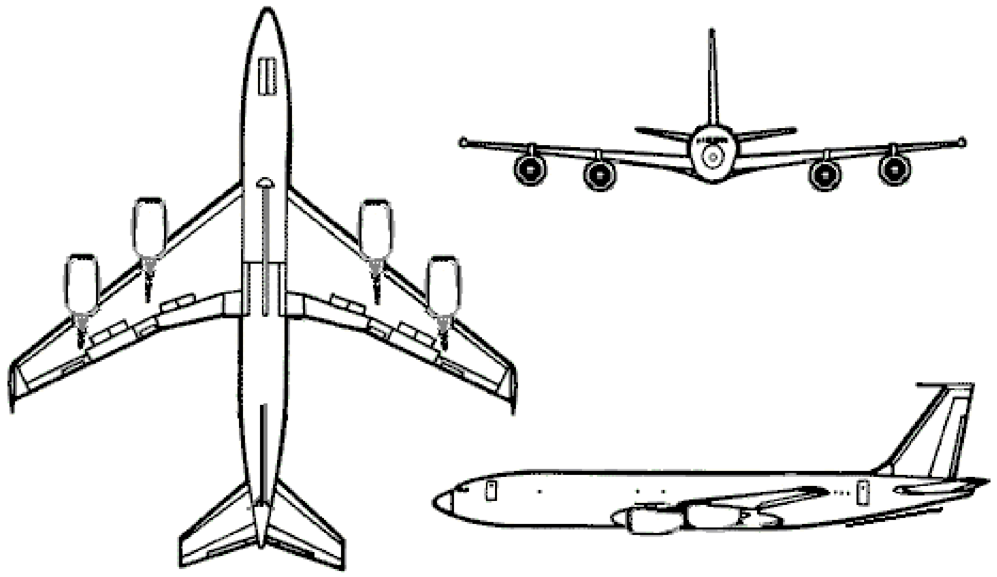


Fig. 2.3: aircraft Tanker model

2.2 Model of the boom

The boom has been modeled using the scheme represented in Fig. 2.4. The boom is connected to the Tanker at point P and consists of two elements: the first element is connected to point *P* by two revolute joints, allowing vertical and lateral relative rotations (θ_4 and θ_5); the second element is connected to the first one by a prismatic joint that allows the extension d_6.
The dynamic model of the boom has been derived using the Lagrange method:

$$\frac{d}{dt}\frac{\partial L(q,\dot{q})}{\partial \dot{q}_i}-\frac{\partial L(q,\dot{q})}{\partial q_i}=F_i, \quad i=1,\ldots,n \tag{2.1}$$

where $L(q,\dot{q})=T(q,\dot{q})-U(q)$ is the Lagrangian (difference between the boom kinetic and potential energy), q is the vector of the Lagrangian coordinates and F_i are the lagrangian forces on the boom.

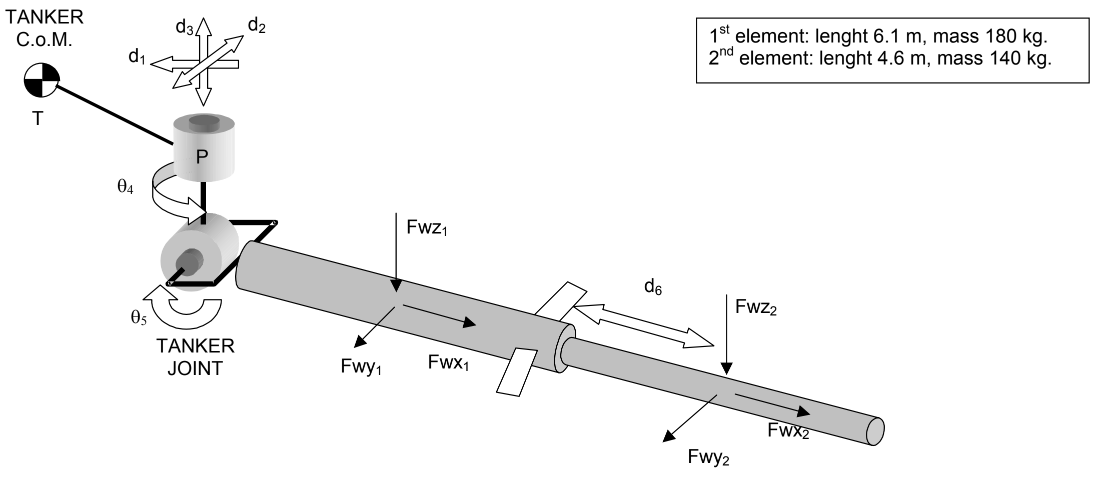

TANKER C.o.M.
T
d3
d2
d1
P
θ4
θ5
TANKER JOINT
Fwz1
Fwy1
Fwx1
d6
Fwz2
Fwx2
Fwy2
1st element: lenght 6.1 m, mass 180 kg.
2nd element: lenght 4.6 m, mass 140 kg.

Fig. 2.4: model of the refueling boom

To derive the Lagrangian, reference is made to an inertial frame, the ERF; in this case the inertial and gravitational forces are implicitly included in the left-hand side of (2.1) and F_i represent the active forces (wind and control forces). With respect to this frame, the boom has six degrees of freedom: the three translations d_1, d_2, and d_3 of point P, the rotations θ_4 and θ_5, and the extension d_6; therefore the Lagrangian coordinates can be chosen as $q = \left[d_1, d_2, d_3, \theta_4, \theta_5, d_6\right]^T$.

Fig. 2.5: Boom operator console

Furthermore, the first three variables d_1, d_2, and d_3 are expressed in function of the tanker position as:

$$\begin{aligned} {}^{E}P(t) &= {}^{E}T(t) + {}^{E}TP \\ {}^{E}\dot{P}(t) &= {}^{E}\dot{T}(t) + \omega \times {}^{E}TP \\ {}^{E}\ddot{P}(t) &= {}^{E}\ddot{T}(t) + \dot{\omega} \times {}^{E}TP + \omega \wedge \left(\omega \wedge {}^{E}TP\right) \end{aligned} \qquad (2.2)$$

where ${}^{E}T$ is the position of the Tanker's center of gravity, ω is the tanker angular velocity, ${}^{E}P = [d_1, d_2, d_3]^T$, ${}^{E}TP$ is the fixed length vector going from ${}^{E}T$ to ${}^{E}P$.

The kinetic and potential energies have be derived referring to the Denavit-Hartenberg representation of the system:

	a_i	α_i	d_i	θ_i
1	0	$\frac{\pi}{2}$	d_1	0
2	0	$\frac{\pi}{2}$	d_2	$\frac{\pi}{2}$
3	0	$\frac{\pi}{2}$	d_3	0
4	0	$-\frac{\pi}{2}$	0	θ_4
5	0	$-\frac{\pi}{2}$	0	θ_5
6	0	$\frac{\pi}{2}$	d_6	$\frac{\pi}{2}$

Tab. 2.1: Denavit-Hartenberg Boom parameter

In the AAR simulation the boom is controlled using a joystick and there is a camera point of view that corresponds to the operator point of view during the refueling maneuver (Fig.2.6).

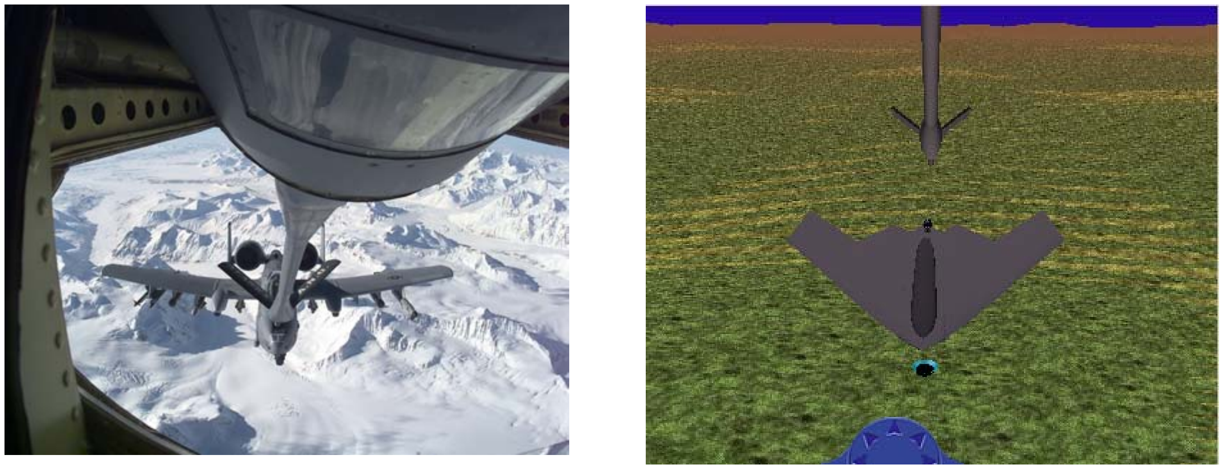


Fig. 2.6: Boom operator view in a real and simulated refueling maneuver

2.3 Markers

The markers are bright red lights placed in the Tanker, they are modeled as spheres with a radius of 10 cm each. In the AAR simulation the markers are 9 and they have a default configuration (Fig 2.7), but there is a possible for modify the number until a maximum of 10 and the position of the markers. In the next chapters we always refer to the default configuration and we will call the markers with the number that identifies it.

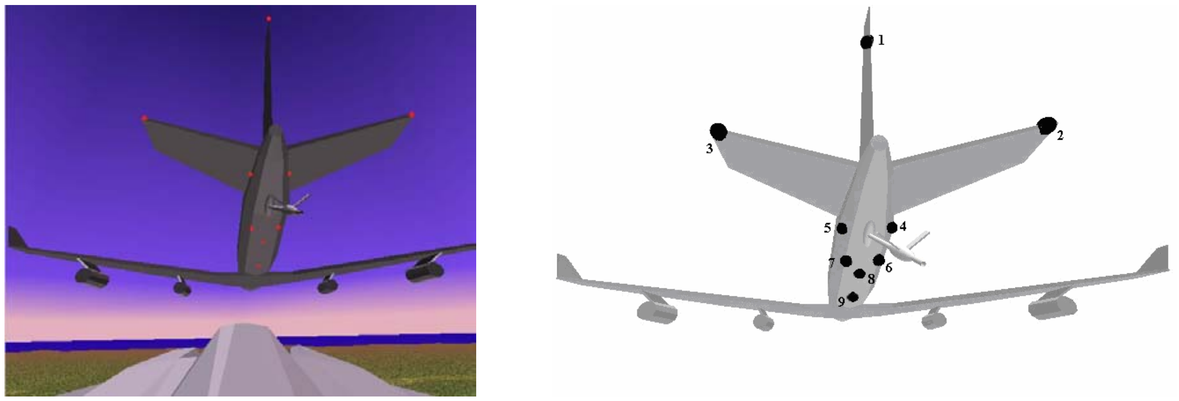


Fig. 2.7: default position of the markers and identification

3. THE UNMANNED AERIAL VEHICLE

Unmanned Aerial Vehicles (UAVs) have been referred to in many ways: RPVs (Remotely Piloted Vehicle), drones, robot planes, and pilot less aircraft are a few such names. Most often called UAVs, they are defined by the Department of Defense (DOD) as powered, aerial vehicles that do not carry a human operator, use aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or non-lethal payload. Ballistic or semi-ballistic vehicles, cruise missiles, and artillery projectiles are not considered UAVs by the DOD definition. UAVs differ from RPVs in that some UAVs can fly autonomously. UAVs are either described as a single air vehicle (with associated surveillance sensors), or a UAV system, which usually consists of three to six air vehicles, a ground control station, and support equipment. UAVs are thought to offer two main advantages over manned aircraft: they are arguably cheaper to produce, and they eliminate the risk to a pilot's life. UAVs protect the lives of pilots by performing the "3-D" missions. Furthermore, for those certain missions which require a very small aircraft, only a UAV can be deployed because there is no equivalent manned system small enough for the job. There are a number of reasons why UAVs have only recently been given a higher priority. Technology is now available that wasn't available just a few years ago, included advanced video surveillance and sensing systems that can be mounted on UAVs.

UAVs range from the size of an insect to that of a commercial airliner. DOD currently possesses five major UAVs: the Air Force's Predator and Global Hawk, the Navy and Marine Corps's Pioneer, and the Army's Hunter and Shadow.

Fig. 3.1: Predator (left) and Pioneer (right) UAVs

The non-military use of UAVs is expected to increase in the future as technologies evolve that allow the safe, reliable flight of UAVs over populated areas. One emerging application is the use of less sophisticated UAVs as aerial camera platforms for the movie-making and entertainment industries. A similar market is growing rapidly in the television news reporting and coverage arenas also. As demand in these markets grows, aircraft such as the IUAS will become a more desirable aerial platform than less-capable hobbyist aircraft, as safety, reliability, ease-of-use, and rapid deployment become important priorities. Additional roles for UAVs in the near future will include homeland security and medical re-supply. The Coast Guard and Border Patrol, parts of the newly formed Department of Homeland Security, already have plans to deploy UAVs to watch coastal waters, patrol the nation's borders, and protect major oil and gas pipelines. Congressional support exists for using UAVs for border security. During a Senate Armed Services Committee hearing on homeland defense, it was stated that although it would not be appropriate or constitutional for the military to patrol the border, domestic agencies using UAVs could carry out this mission. On the medical side, UAVs such as the Army's Shadow have been studied as delivery vehicles for critical medical supplies needed on the battlefield. Not all of these new applications have been approved — UAV advocates state that in order for UAVs to take an active role in homeland security, Federal Aviation Administration (FAA) regulations concerning the use of UAVs will have to change. The Coast Guard will most likely take the lead in resolving UAV airspace issues with the FAA. The National Aeronautics and Space Administration (NASA) and the UAV industry will also be working with the FAA on the issue, as they are joining forces in an initiative to achieve routine UAV operations in the national airspace within a few years.

3.1 Model of used UAV

The aircraft model used in AAR simulation is an ICE-101 [33], the model has been developed using the conventional modeling approach outlined in [32]. The resulting UAV model is described by a 12 steady state model:

$$x = \left[V, \alpha, \beta, p, q, r, \psi, \theta, \varphi, {}^{E}x, {}^{E}y, H\right] \tag{3.1}$$

where x is the state variable; V (m/s) is the component x of the velocity in body axis; α (rad) is the wind axis angle of attack; β (rad) is the wind axis sideslip angle; p, q, r (rad/sec) are the components (x, y, z) of the angular velocity in body axis (also known as roll, pitch and yaw rates); ψ, θ, φ (rad) are the yaw, pitch and roll Euler angles; ${}^{E}x$, ${}^{E}y$, H are the position in Earth fixed Reference Frame (ERF).
The angle of attack α and the sideslip angle β are defined as:

$$\alpha = \tan^{-1}\left(\frac{W}{\left\|\bar{V}\right\|}\right) \quad \text{and} \quad \beta = \sin^{-1}\left(\frac{U}{\left\|\bar{V}\right\|}\right) \tag{3.2}$$

where $\bar{V} = \left[V, U, W\right]$ is the linear velocity in body axis.

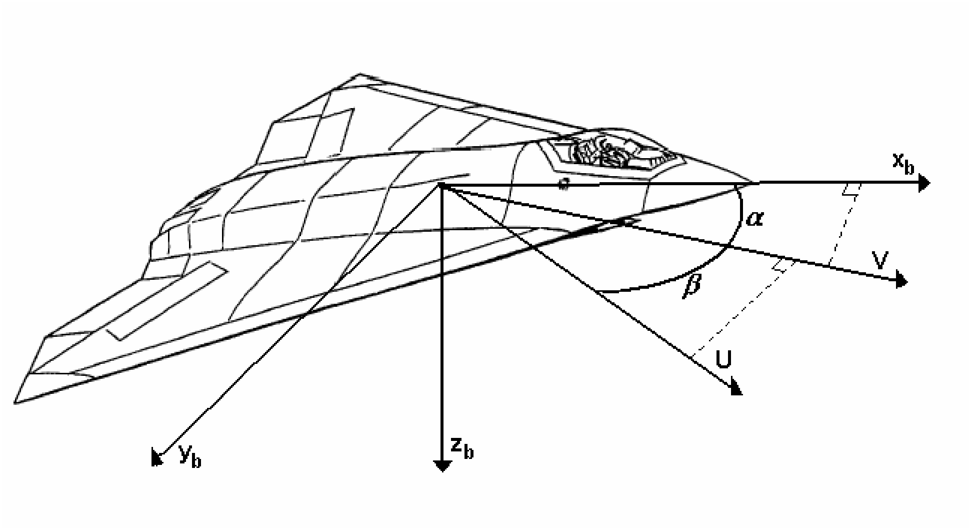


Fig. 3.2: angle of attack α and sideslip angle β definition

The input vector u is:

$$u = \left[\delta_{Throttle}, \delta_{AMT\_R}, \delta_{AMT\_L}, \delta_{TEF\_R}, \delta_{TEF\_L}, \delta_{LEF\_R}, \delta_{LEF\_L}, \delta_{PF}, \delta_{SSD\_R}, \delta_{SSD\_L}\right] \tag{3.3}$$

where AMT is All Moving Tips, TEF is Trailing Edge Flaps, LEF is Leading Edge Flaps, PF is Pitch Flaps, SSD is Spoiler Slot Deflector, the position of the control surface are shown in Fig. 3.3.

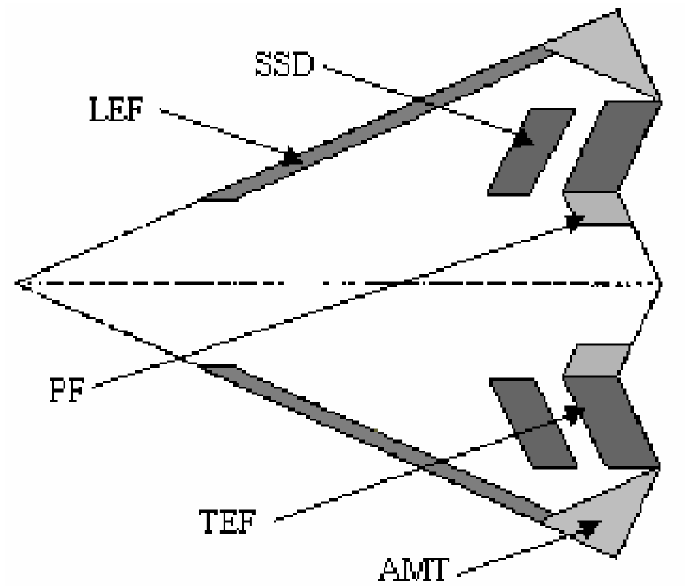


Fig. 3.3: Position of control surface

The dynamic UAV model can be described by the differential equation

$$\dot{x}(t) = f(x,u,t) \tag{3.4}$$

which can be linearized in a trim point such as:

$$\dot{x}(t) = f(x_0,u_0,t) = 0 \tag{3.5}$$

in the above condition, the UAV has acceleration equal to zero and module of the vector velocity constant. The equivalent state space model:

$$\dot{x} = Ax + Bu \tag{3.6}$$

is an LTI system with 12 state variable, the analysis of (3.6) shown that there is a substantial decoupling between the longitudinal symmetric motion (translation *x*, translation *z* and rotation *y*) and the lateral-directional asymmetric motion (translation *y*, rotation *x* and rotation *z*).

The longitudinal motion is characterized by 2 modes, the first one with high frequency (short period) dominated by negligible variation in velocity, the second one with low frequency (phugoid) characterized by small variation of incidence and slow variation on the pitch angles:

$$\begin{aligned} \lambda_1 &= \lambda_{1,R} \pm j\lambda_{1,I} \quad (short\ period) \\ \lambda_2 &= \lambda_{2,R} \pm j\lambda_{2,I} \quad (phugoid) \end{aligned} \tag{3.7}$$

The lateral-directional motion is characterized by 4 real modes, of which 2 are unstable:

$$\begin{aligned} &\lambda_3 > 0 \\ &\lambda_4 < 0 \\ &\lambda_5 < 0 \\ &\lambda_6 > 0 \end{aligned} \tag{3.8}$$

for this reason the model results with very unstable lateral dynamics.

3.2 Sensors

The UAVs need more sensors of a normal airplane. Normally the precision in the sensors play a major role in the performance of a system, in a UAV this is even more important since there is no human intervention during the flight phases.

It is assumed that the UAV has a GPS system, an Inertial Navigation Unit (INU) with gyros and accelerometers and a Machine Vision (MV) system.

The INU provides ψ, θ, φ (rad), and p, q, r (rad/sec),with some added noise. In this effort, a Band-limited White Gaussian Noise (BWGN) with a power of (n_p) = $1*10^{-9}$ and sample time T =0.05 sec is assumed for the euler angles, and an BWGN with a n_p = $1*10^{-8}$ and T=0.05 sec, is assumed for p, q, and r . The measurements of the velocity angles α and β (rad) have a n_p = $1*10^{-9}$ and T=0.05 sec, and the velocity V (m/s) has a n_p = $1*10^{-7}$ and T=0.05 sec. The accelerations and the other measurements do not have any added WGN.

The GPS system provides ${}^{E}x$, ${}^{E}y$, H , for which a WGN with n_p = $1*10^{-3}$ and T=0.1 sec is assumed. Furthermore, the GPS has a unit delay to better approximate the real behavior of this system. The simulation of these sensors is shown in Fig 3.4.

A BWGN is a simulation of White Gaussian Noise with the high of the Power Spectral Density (PSD) equal to n_p, the correlation time is equal to the sample time T and the covariance is $c = \frac{n_p}{T}$.

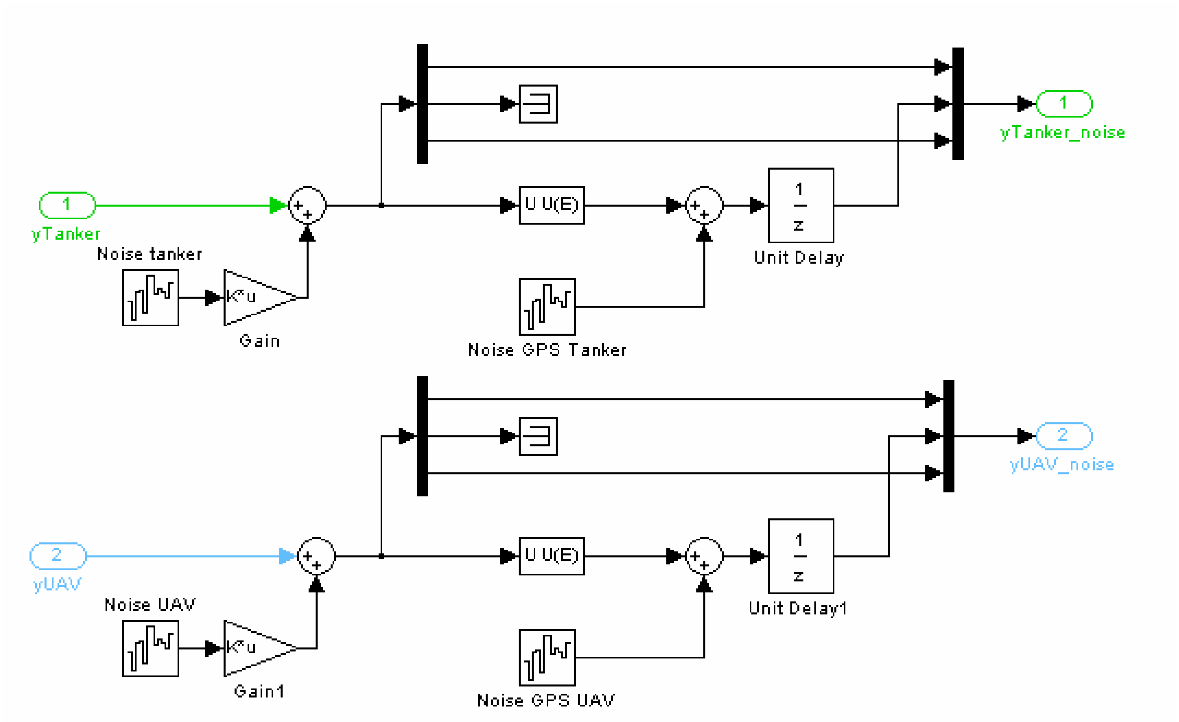


Fig. 3.4: Sensor block scheme

The MV system can be consider a smart sensor, that provides the distance between camera and observed point (Tanker). Within this effort, the MV system requires that the tanker has some bright markers, placed in a known positions. It is nevertheless possible to use feature extraction algorithms that could detect tanker corners in known positions, thereby avoiding the pacement of bright markers on the tanker. The MV system is described in detail in the following chapter of this document.

The availability of a communication channel between UAV and Tanker is crucial, since the UAV system has to know the position and orientation of the Tanker in order for a GPS-only based system to work correctly. It is important to notice that the evaluation of the distance requires not only the measurements coming from the GPS and MV but also the measurements of the attitude angles of the UAV and the Tanker. Furthermore, as outlined below, the AAR control laws require the measurement of all the UAV states. The measurement process was modeled by corrupting the signals with an additive zero mean random noise with a typical STD. Transmission and processing delay have been considered in the AAR simulation. As for the transmission delay, 55 data bytes have to be transmitted: 13 floating point numbers (12 state variables and the time, 4 byte/number), plus header and checksum (3 bytes). Assuming a bit rate of 19.6 kbits/s, the transmission time is about 55x8/19600 ≈ 0.023 ms. As for the processing time, the algorithm runs in about 0.02 s on the Simulink environment (no accelerator) on a Pentium 4 2.6GHz Windows 2000 Operating System. Therefore the estimation of 0.05 s for the overall delay was assumed.

Thanks to these sensors we have been able to set a method of fusion between GPS and MV systems. The basic idea is that the measurement is entirely provided by the GPS (d_{GPS}) when the UAV has a distance d from the Tanker greater or equal to d_1 and it is entirely provided by the MV (d_{MV}) if the distance UAV - Tanker is lesser or equal to d_2. If $d_1 < d < d_2$ we have a linear interpolation between the distance provided by the MV and that provided by the GPS, with the rule:

$$d_{GPS}\left(\frac{d-d_2}{d_1-d_2}\right)+d_{MV}\left(1-\frac{d-d_2}{d_1-d_2}\right) \tag{3.9}$$

the fusion system is presented in Fig 3.5.

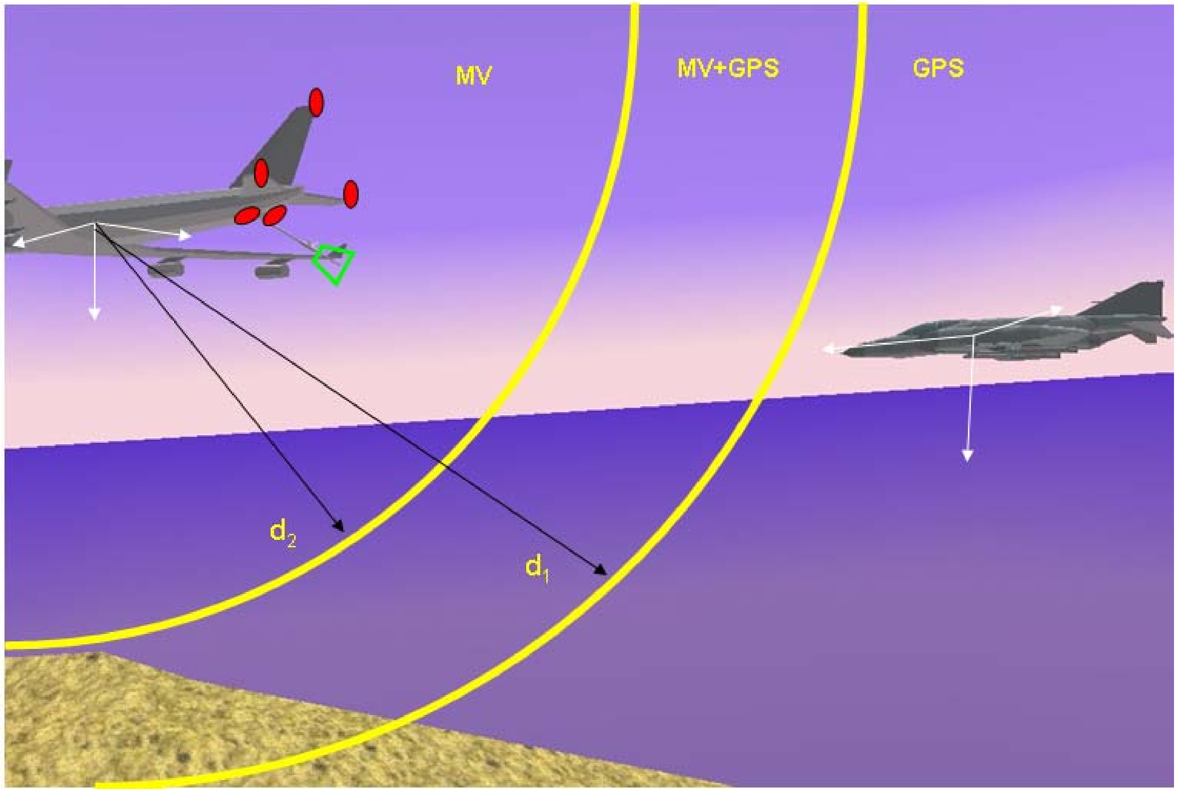


Fig. 3.5: fusion between GPS and MV systems

In more detail, the data provided by the GPS and INU is used to calculate the homogeneous transformation matrix ${}^{C}T_{T}$, which is the matrix that transforms a vector with origin in TRF in a vector with the origin in CRF; a calculation method for this matrix is presented in Fig 3.6. ${}^{C}T_{T}$ is also the output of the MV system, as can be seen in Fig 3.7, the other output of the MV is the number of markers (US_MK) used for the current estimation. At this point we can join the two measurements with the fusion method previously described. The MV measurement is valid only is the current measurement is provided with at least 5 markers. Therefore, the fusion is performed if the MV has a number greater or equal to 5 in US_MK and only for the translation vector (x, y, z) of the matrix ${}^{C}T_{T}$. The Euler angles provided by the MV tend to be more noisy than the ones provided by the MV system, furthermore, the latter are in principle always available independently on external conditions such as the number of detected markers. We choose, for that reason, to use the angles that come from INU in the feedback loop.

The output of the fusion block is fed back as input of the MV block because this is deemed to be the "best estimation" for the value of ${}^{C}T_{T}$ at the next sampling time.

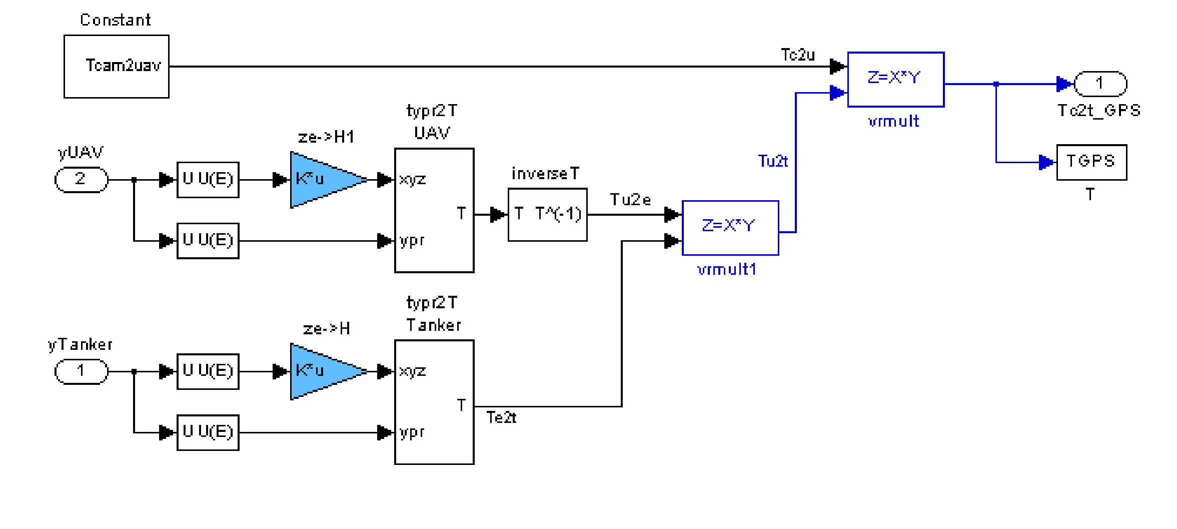


Fig. 3.6: calculus of ${}^{C}T_{T}$

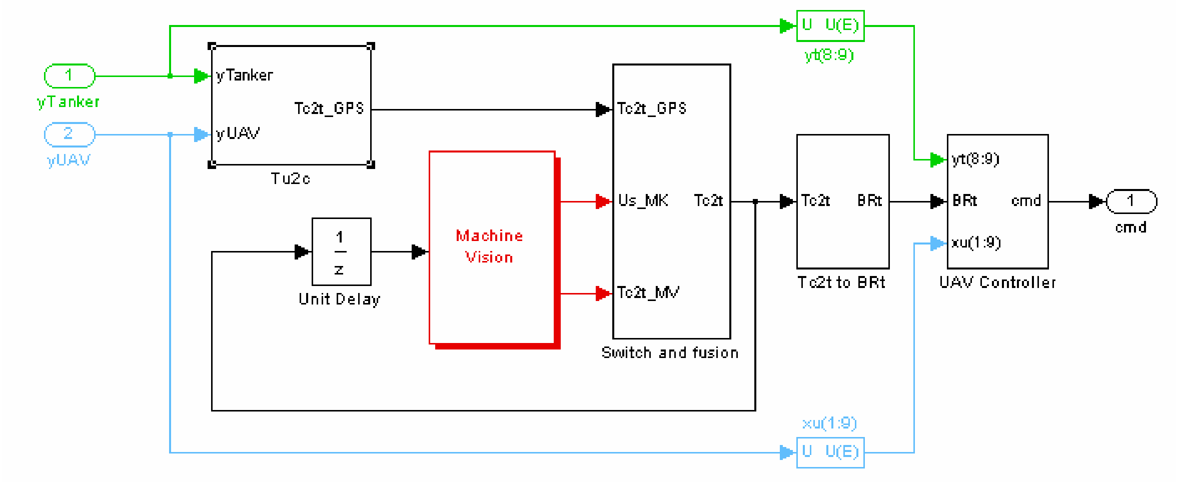


Fig. 3.7: UAV software scheme

3.3 Control

The controller has to be able to maintain the aircraft on a defined trajectory, that is the UAV controller has to preserve the internal stability (the model is dynamically unstable in the lateral-directional motion), has to follow a docking path, and finally keep the distance between Receptacle point (R) on the UAV and Box Point (B) on the Tanker as low as possible. This kind of minimization problem is typical in the control theory and can be resolved using a Linear Quadratic Regulator (LQR) controller. The LQR control tries to minimize a performance cost function *J* which depends quadratically on the output vector *Y* and the input *U*.

In this problem we can define an augmented state vector:

$$X_{AUG} = \left[V, \alpha, \beta, p, q, r, \psi, \theta, \varphi, e_x, e_y, e_z, \int e_x, \int e_y, \int e_z\right] \tag{3.10}$$

where e_x, e_y and e_z are the *x, y* and *z* distance between the point R and the point B and the reference trajectory. The input vector *U* is the same defined in (3.3). At this point we can define a performance cost function:

$$J = \int_0^\infty \left(Y^T Q Y + U^T R U\right) dt \tag{3.11}$$

where Q and R are diagonal matrices that establish the performance for each used variable, the *Y* variable is:

$$Y = \left[e_x, e_y, e_z, \int e_x, \int e_y, \int e_z\right] \tag{3.12}$$

In the augmented state vector X_{AUG} there is the integral of the distances in order to guarantee that the distances converge to zero.

The controller uses the nine state variables of the UAV system, the vector BR expressed in Tanker frame (TRF), and the pitch and roll angles of the Tanker (Fig 3.7) to provide one vector *cmd* of 11 elements. It can be noticed that (Fig 3.8) the vector BRt is rotated of the angles pitch and roll of the Tanker to yeald the vector BR in a reference frame called Psi Frame (MRF). MRF has the same origin of ERF but is oriented on the direction of the Tanker; this allows to control the aircraft as if it was always directed toward north in ERF. The nine state variables are subtracted to the trim point x_0. The reference trajectory, generated as a cubic path in three dimensions, is subtracted to the BR vector in MRF. Once the integrals of e_x, e_y and e_z are available, we can find the input vector *U* for the state space model using the formula:

$$U = K_{LQR} X_{AUG} \tag{3.13}$$

where K_{LQR} represent the MIMO static controller with 15 inputs and 11 outputs.

The variable *cmd* is the sum of *U* and the trim input constant. In this way non-linear systems are locally controlled through a simple linear controller in a trim point.

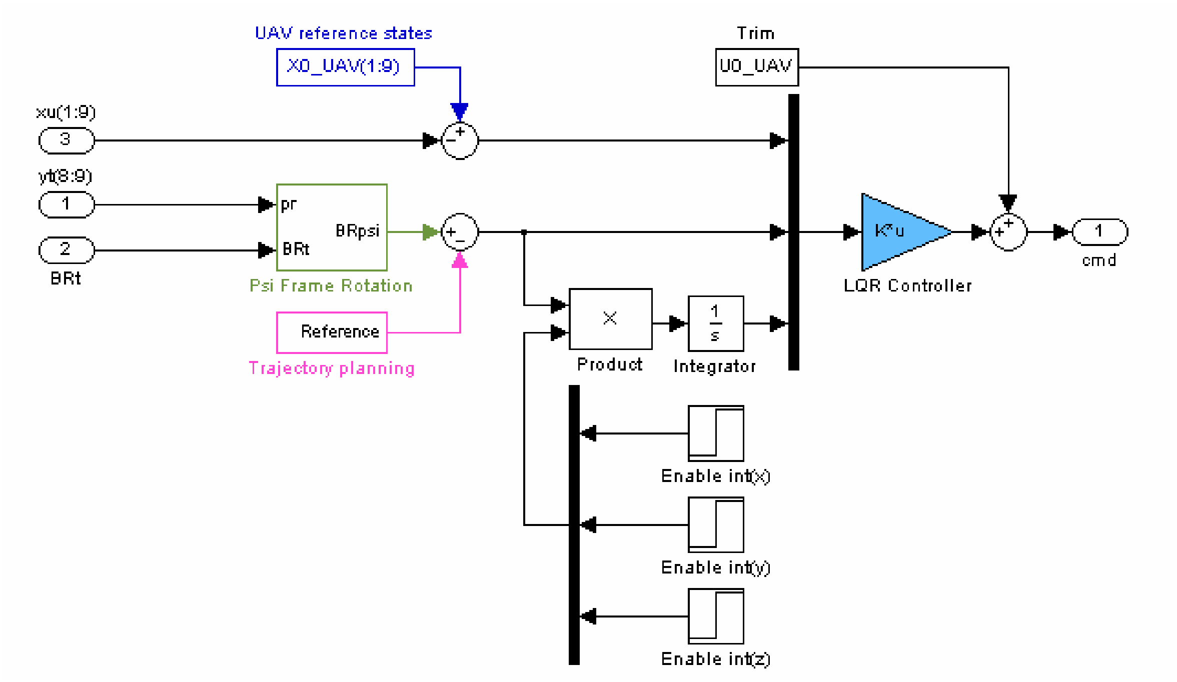


Fig. 3.8: Controller scheme

3.4 Atmospheric turbulence and wake effect

The atmospheric turbulence acting on the probe system and on both Tanker and the UAV aircraft has been modeled using the Dryden wind turbulence model [34]. A ‘light’ turbulence was selected since aerial refueling is typically performed at high altitudes in calm air [34]. The wake effects of the Tanker on the UAV are more significant than the atmospheric turbulence and have been modeled through the interpolation from a large amount of experimental data [35][36] as perturbations to the aerodynamic coefficients $C_D, C_L, C_m, C_l, C_n, C_Y$ for the UAV aerodynamic forces and moments. These coefficients represents: Drag and Lift coefficient, rolling, pitching and yawing moment coefficient and Side Force coefficient, all of them are subject to variations due to formation flight.

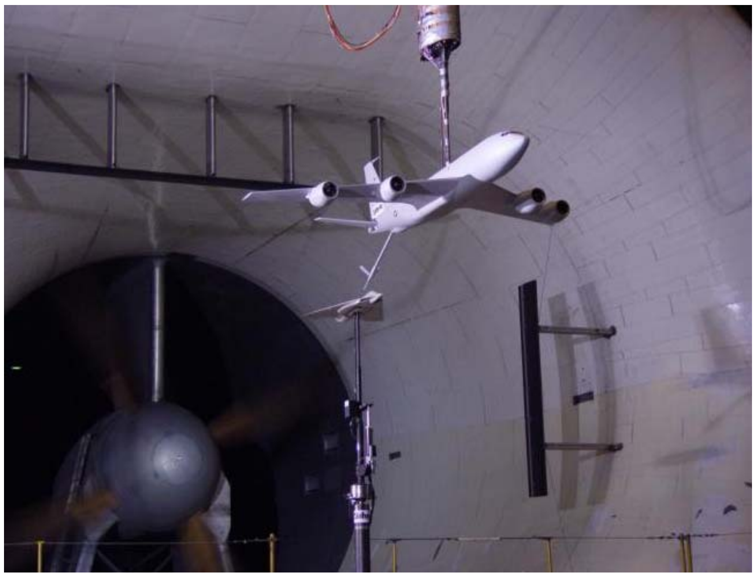


Fig. 3.9: AAR test to search the coefficient increment

3.5 Actuators dynamics

To have a more realistic implementation of the aircraft modeling of the actuators has been added. Every actuator of the input vector (3.3) has a saturation and a rate limiter (Fig, 3.10) that is, an upper and lower limit and a velocity limit. The input vector is also delayed, and every actuator is filtered, the filters are faster for the control surfaces than for the throttle command.

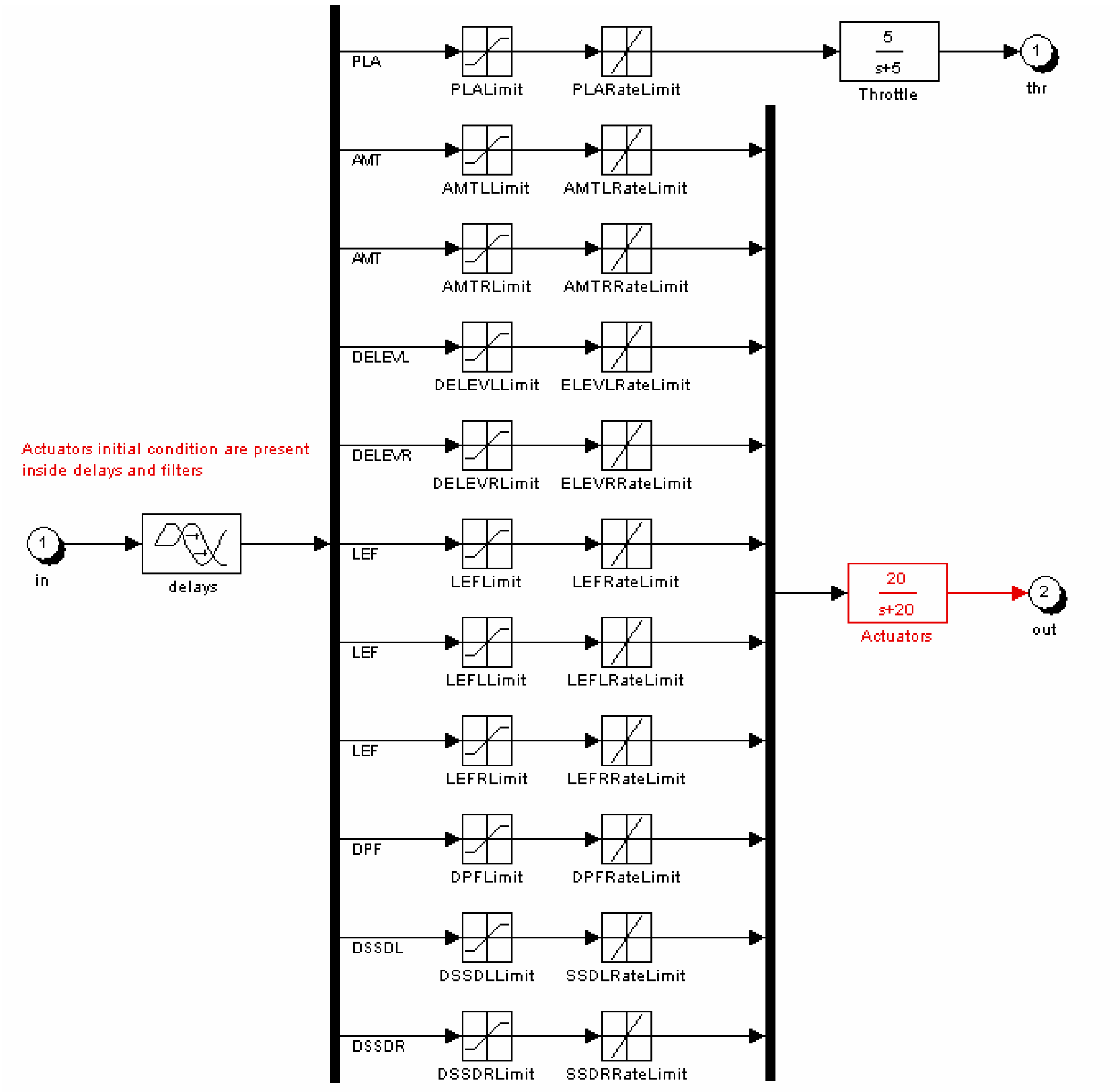


Fig. 3.10: actuators dynamics

4. THE MACHINE VISION SYSTEM

Active sensors based on structured lighting are now compact and fast enough to be used in visual servoing [27][29]. In addition to the specific class of sensors and markers, the performance of a Machine Vision (MV) system depends on the performance of the marker detection, labeling and pose estimation methods. In a MV system is very important that sampling time is proportionate to the relative dynamics of the airplanes. Different approaches for the marker detection/labeling are outlined in Refs [2] [30] [31]. Within this effort the attention is focused on the labeling algorithm for the detected markers and the pose estimation problem [4] with a variable or fixed number of visible markers. Thus, for our purposes it reasonable to describe the markers as points in space and to model the image formation process by prospective projections [29] , this method is also known as "pin hole" model [27] and is detailed later.

4.1 The pin hole model

The pin hole model is the simplest geometric model for image construction. Let *P* be a scene point with coordinate *(X, Y, Z*), and let *P'* the projection on image plane, with coordinates *(X',Y',Z')*. If *f* is the distance between the hole *O* and the image plane (*focal length*), for the similarity of the triangle we have:

$$-\frac{Y'}{f}=\frac{Y}{X} \quad \text{and} \quad -\frac{Z'}{f}=\frac{Z}{X} \tag{4.1}$$

therefore

$$Y'=-f\frac{Y}{X} \;,\quad Z'=-f\frac{Z}{X} \;,\quad X'=-f \tag{4.2}$$

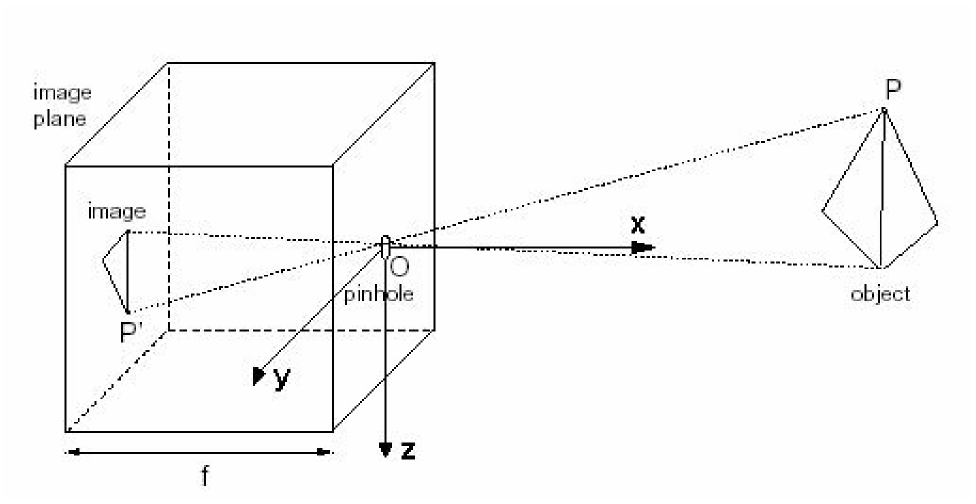


Fig. 4.1: geometry of image construction in pin hole camera

Note that the image is inverted in comparison to the scene both right-left that up-down, in the (4.2) there is minus sign. These equations define the image process formation that is known as *perspective projection*. Subsequently we call (u_j, v_j) the perspective projection of the point $P_j = (X_j, Y_j, Z_j)$.

4.2 The Machine Vision block

In the AAR simulation considered there is a block that manages the classical MV function, this block correspond to a smart sensor that provides the measurement with a sampling time of 0.1 *sec*. We can see in Fig 4.2 the composition of this block.

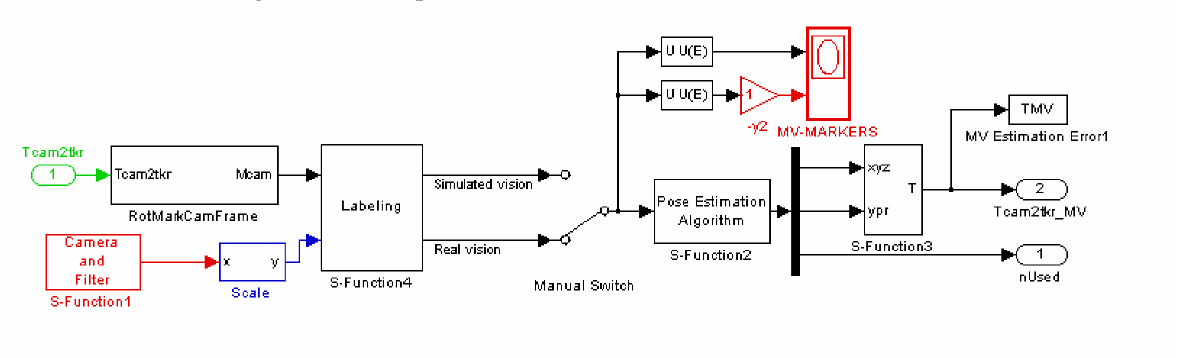


Fig. 4.2: MV scheme

The MV has one input port (${}^{C}T_{T}$) that is the homogeneous transformation matrix (4 x 4) from Tanker Reference Frame to Camera Reference Frame, which is composed as follows:

$$
{}^{C}T_{T} = \left[\begin{array}{ccc|c} & & & \\ & {}^{C}R_{T} & & {}^{T}TC \\ & & & \\ - & - & - & - \\ 0 & 0 & 0 & 1 \end{array}\right] \tag{4.3}
$$

where ${}^{C}R_{T}$ is the rotation matrix that changes the reference frame from Tanker frame to Camera Reference Frame, and ${}^{T}TC$ is the translation vector (*x*, *y*, *z*) that has the origin in the Tanker Center of Gravity (CG_{TK})and the end in the Camera origin. The matrix ${}^{C}T_{T}$ is used to transform a vector expressed within TRF in a corresponding vector in CRF. This matrix contains all the information necessary to express the relationship among two different reference system, which in our case are the Tanker frame and Camera frame.

The MV block has two outputs, the first one (*nUsed*) is the number of used markers in the pose estimation problem, and the second one (${}^{C}T_{T}$) is the homogeneous transformation matrix from Tanker to Camera composed as previously shown and provided by the Pose estimation Algorithm.

4.2.1 Camera and Filter

The Camera and Filter is a Matlab® S-Function that captures the image that Virtual Reality Toolbox® (VRT®) shows on the screen. The image has dimensions (320 x 200), and after the capture this is mapped into the memory as a matrix (320 x 200 x 3) where the third dimension represented the intensities of the color red, green and blue. We know that the markers are red, for this reason we convert only the sub-matrix with the red intensity, using an appropriate threshold to transform the sub-matrix into a black and white (BW) image. Using standard morphological filtering techniques, we search into the BW image all the connected objects and extract the centroid coordinate for each connected object. The output of the function is a vector with all the coordinates (u'_{j}, v'_{j}) of all connected objects found, often, these object are simply the markers, but sometimes, some points that are also connected objects but are not markers, show up in the output.

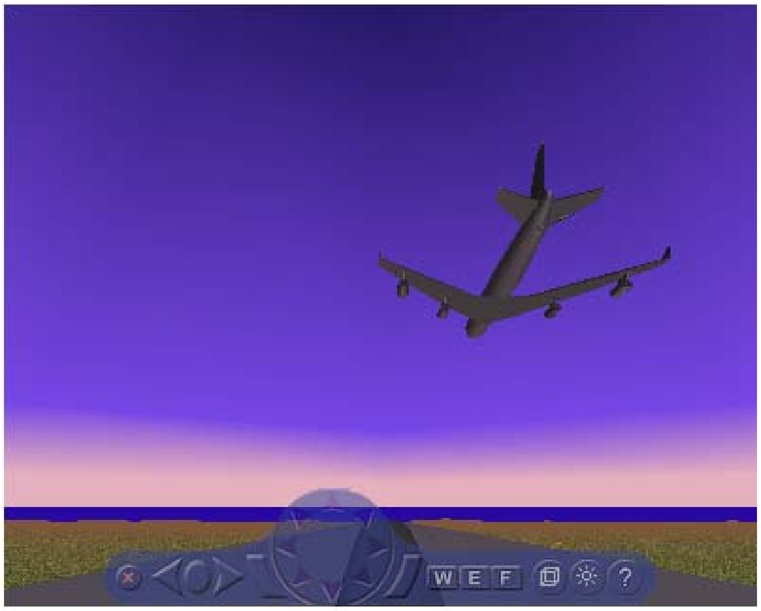


Fig. 4.3: image captured from VRT®

4.2.2 Scale

The scale function transforms the 2D coordinates of the detected markers from camera plane (u'_j, v'_j) express in pixel into Camera Reference Frame (u_j, v_j) express in meters, known the vertical and horizontal dimension of one pixel and the dimension of the screen

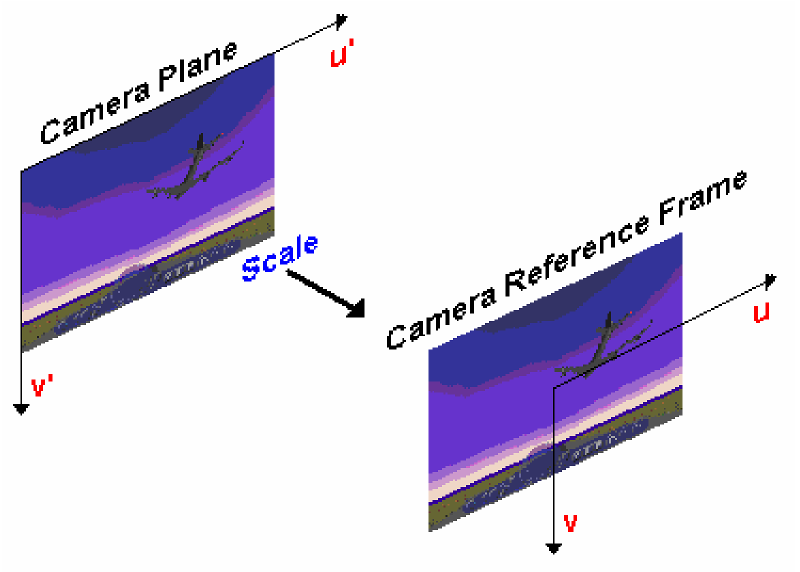


Fig. 4.4: the scale function

4.2.3 Rotation markers in camera frame

The block *RotMarkCamFrame* in Fig 4.2 is open and shown in Fig 4.5. This block provides an estimation of the markers position in 3D Camera Reference Frame (CRF). If we pre-multiply the matrix ${}^{T}M$ of the markers points expressed within the Tanker Reference Frame (TRF), by the rotation matrix ${}^{C}T_{T}$, we obtain the actual position of the markers within CRF ${}^{C}M$, the last operation consists in bringing the matrix in classical 3D coordinates.

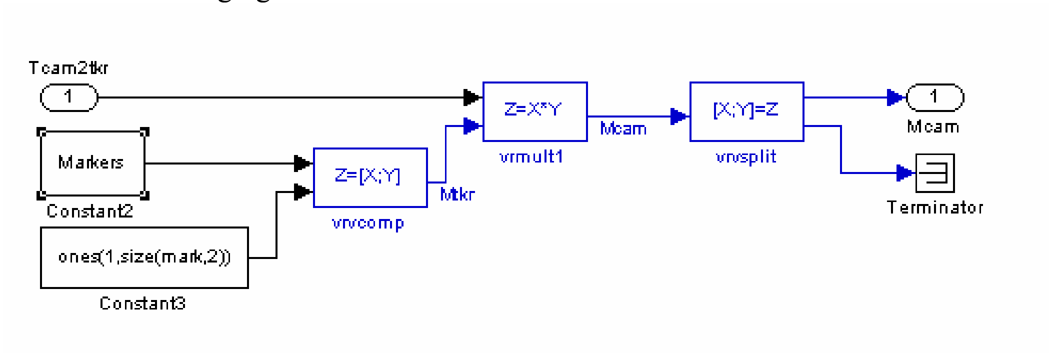


Fig. 4.5: rotation markers in camera frame scheme

4.2.4 The Labeling function

In a MV system the labeling of the detected markers is one of most important and complicated parts. Initially, the set of points $p_j \equiv [u_j, v_j]$ from the camera measurements are not related to the actual markers on the tanker. The problem can be formalized in terms of matching the set of points $\overline{P} = (p_1, p_2, ..., p_m)$ to the set of points $\hat{\overline{P}} = (\hat{p}_1, \hat{p}_2, ..., \hat{p}_n)$ where $\hat{p}_j \equiv [\hat{u}_j, \hat{v}_j]$. In this effort the set $\overline{P}$ represents the set of the *m* 'to be matched' detected markers extracted by the camera measurements, while the set $\hat{\overline{P}}$ represents the set of the *n (n=9)* "nominal" markers estimated. Since the data sets $\overline{P}$ and $\hat{\overline{P}}$ represents the 2D projections of the same markers at the same time instant on the same plane (as shown in Fig. 4.6) a high degree of correlation between the two sets is expected. In the ideal case corresponding points would be exactly superimposed, resulting in a trivial matching process. However, in the presence of different sources of system and measurement noise, a matching problem has to be defined and solved.

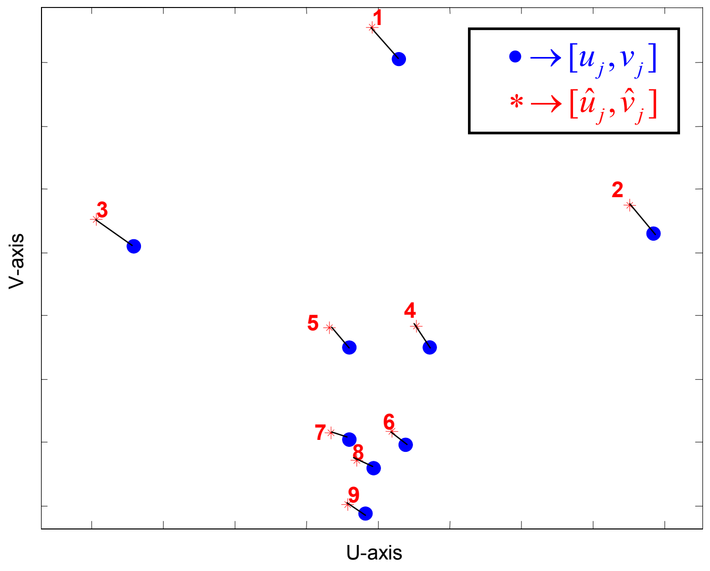


Fig. 4.6: Matching between the labeled set of points $\hat{\overline{P}}$ and the unlabelled set $\overline{P}$.

A detailed technical literature describes a number of robust matching techniques for point sets [45]. Usually, the degree of similarity between two data sets is defined in terms of a cost function or a distance function derived on general principles as geometric proximity, rigidity, and exclusion [46]. The best matching is then evaluated as the result of on optimization process exploring the space of the potential solutions. Often, the problem can be set as a classical assignment problem, and therefore solved using standard polynomial Network Flow algorithms. A definition of the point-matching problem as an assignment problem, as well as an extensive analysis of different labeling algorithms, in a different, still to be published, study [47]. In this effort, the author has chosen to move beyond the simple assignment problem paradigm, since it was deemed too restrictive to correctly represent the point matching problem in real world situations. For example, it often happens that, due to misleading visual clues, points that are not markers are interpreted as such from the image processing algorithms, leading to situations where the set $\overline{P}$ contains more elements than $\hat{\overline{P}}$. In these cases, an algorithm that just tries to minimize a linear function of the involved distances (thereby solving a simple LP assignment problem) usually performs poorly. Conversely, an algorithm that instead has the ability to deliberately ignore some markers by considering them "erroneous" (thereby solving a problem that cannot be cast as a simple assignment one) could provide better performance.

The developed labeling function has two inputs: the first one is an estimation of the markers position in the 3D CRF (the output of the block analyzed in section 4.2.3), the second one is the vector of detected markers provided by the camera and scaled (the output of the block analyzed in section 4.2.2).

The labeling function has two outputs: the first one, called Simulated Vision (SV), provides a vector $[u_1,\ v_1,\ \ldots\ldots,\ u_n,\ v_n]$, which is simply the projection of the estimated markers position in the 2D CRF using the rule (4.2) of the pin hole camera. The second output, called Real Vision (RV), provides the vector for which the labeling function has been really created, that is an ordered vector of detected markers.

The labeling function, therefore, has the purpose to detect the points that correspond to real markers and arrange the output vector so that it has the format $[u_1,\ v_1,\ \ldots\ldots,\ u_n,\ v_n]$. If the k^{th} marker is not detectable the overflow value 100 is used instead in the position *2* k* and *2*k+1*.

Now we explain how the function works: let M denote the set of the *n* physical markers, and let $\overline{M}$ denote the set of detected markers (not to exceed *m*). The labeling function creates a matrix Err of dimension *n*-by-*m*, whose coefficients are all the Euclidian distance between the markers M and $\overline{M}$. Three vectors, $MinR$, $MinC$ and $Index$ - with dimensions *n*, *m* and *m* respectively - are also created, as shown in Fig. 4.6. The minimum element of the column is stored in the row vector $MinC$ while the minimum element of the row is stored in the column vector $MinR$. The index of the row in which the function finds the minimum is stored in the row vector $Index$. The position of one detected marker $\overline{M}$ is valid if the position j of $MinC[j]$ is equal to $MinR[Index[j]]$, that is the position of $MinR$ indicated by the vector $Index$ in the position j. Using a C syntax the validity condition is given by:

$$MinC[j] == MinR[Index[j]] \tag{4.4}$$

This method works with the hypothesis that the MV system has frequency of 10 Hz, and the aircrafts have low relative dynamics, this two assumptions essentially imply that the positions of the airplanes does not change in meaningful way between the current cycle and the following one. This function, which has a computational complexity of $O(n^2)$, avoids the typical errors associated with a labeling function that simply assigns the detected markers $\bar{M}$ to the nearest markers M .

Err matrix ***MinR vector***

Dist($M_1\bar{M}_1$)	Dist($M_1\bar{M}_2$)	Dist($M_1\bar{M}_3$)	Dist($M_1\bar{M}_4$)	Dist($M_1\bar{M}_5$)
Dist($M_2\bar{M}_1$)	Dist($M_2\bar{M}_2$)	Dist($M_2\bar{M}_3$)	Dist($M_2\bar{M}_4$)	Dist($M_2\bar{M}_5$)
Dist($M_3\bar{M}_1$)	Dist($M_3\bar{M}_2$)	Dist($M_3\bar{M}_3$)	Dist($M_3\bar{M}_4$)	Dist($M_3\bar{M}_5$)

Min(M_1)
Min(M_2)
Min(M_3)

Min($\bar{M}_1$)	Min($\bar{M}_2$)	Min($\bar{M}_3$)	Min($\bar{M}_4$)	Min($\bar{M}_5$)	***MinC vector***
Index(M)	Index(M)	Index(M)	Index(M)	Index(M)	***Index vector***

Fig. 4.7: labeling method for detected markers

For example, typical matching errors may occur if one detected marker is confused with 2 close markers, as shown in Fig. 4.7, where the detected markers *p* and *q* are obviously the same marker but, due a filtering error, two distinct markers are detected instead. At this point, a simple labeling function pairing up the nearest markers would assign '*p*' to '*a*' and '*b*' to '*q*', leading to an incorrect pose estimation. Instead, the developed labeling function avoids the incorrect matching; in fact, while it still assigns '*p*' to '*a*', the fact that '*q*' is closer to the point '*a*' - which has already been assigned - than to the point '*b*' leads to the fact that the condition in (4.4) is false. In turn this leads to discarding the point '*q*' so that at the end only the points that are most relevant for pose estimation purposes are used.

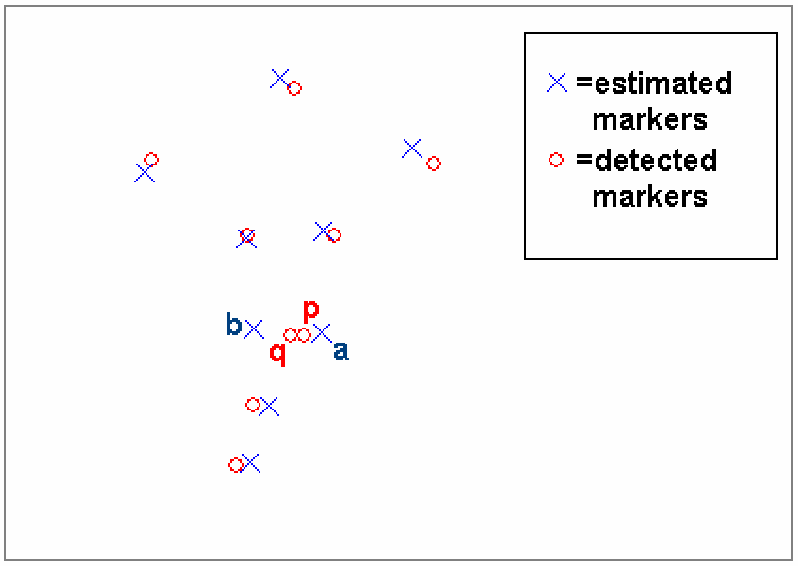
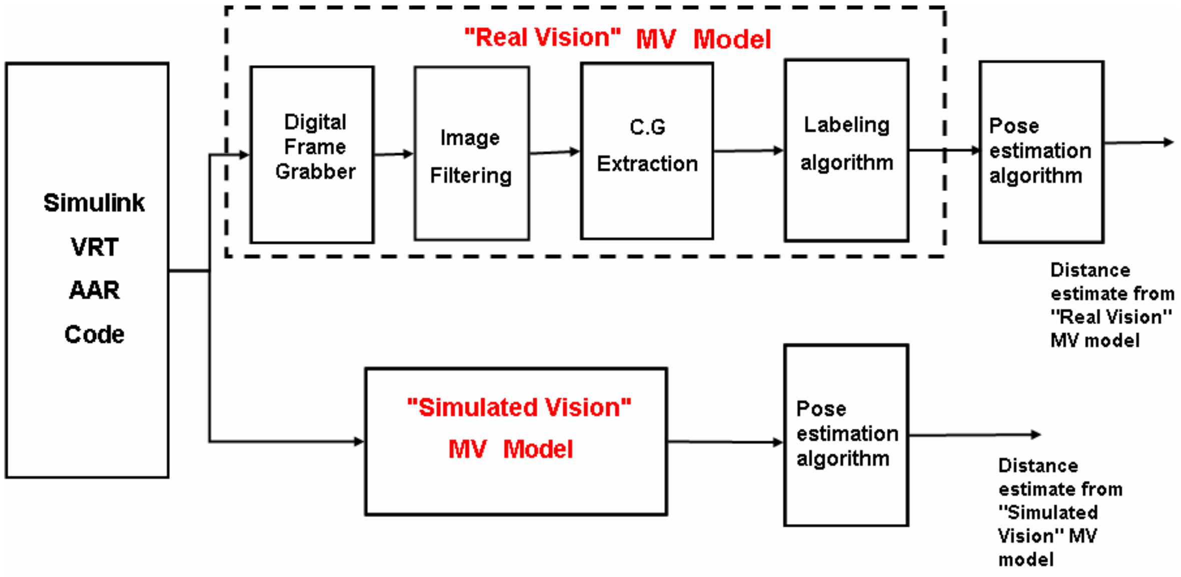


Fig. 4.8: example of avoided error in labeling function

4.2.5 Simulated Vision and Real Vision

Into the MV diagram there is a switch that allows the user to choose between Simulated Vision (SV) and Real Vision (RV), the position of this switch determine which data are provided to the pose estimation algorithm. If we choose the SV, we decide that the pose estimation will be made with the 2D projection of the markers obtained with the (4.2) from the pinhole camera model. The data in SV mode are in every moment complete of each marker position, they are smooth and regular. The SV mode has been created for testing the pose estimation algorithm and uses the MV system without being bound to the performance of the labeling function. The goodness of the pose estimation is clearly dependent by the input of the MV block, in theory we should have that the output matrix ${}^{C}T_{T}$ results equal to the input matrix ${}^{C}T_{T}$. The SV allows, also, using the MV from great distances even if it reduces the MV to a fictitious system.

The Real Vision mode allows one to use the data that come from the camera, this data are certainly noisy and often incomplete since often some markers are not detectable, and sometimes they are not good for the pose estimation algorithm (we will analyze this case in chapter 7), the data depend also by the camera, the filter, the labeling algorithm (see in 4.2.4 the case where the labeling function discard some marker). The RV mode allows the MV system to work with data similar to real-world ones, which is the main reason for which this mode has been developed.



"Real Vision" MV Model
Simulink VRT AAR Code
Digital Frame Grabber
Image Filtering
C.G Extraction
Labeling algorithm
Pose estimation algorithm
Distance estimate from "Real Vision" MV model
"Simulated Vision" MV Model
Pose estimation algorithm
Distance estimate from "Simulated Vision" MV model

Fig. 4.9: block diagram of Simulated Vision and Real Vision

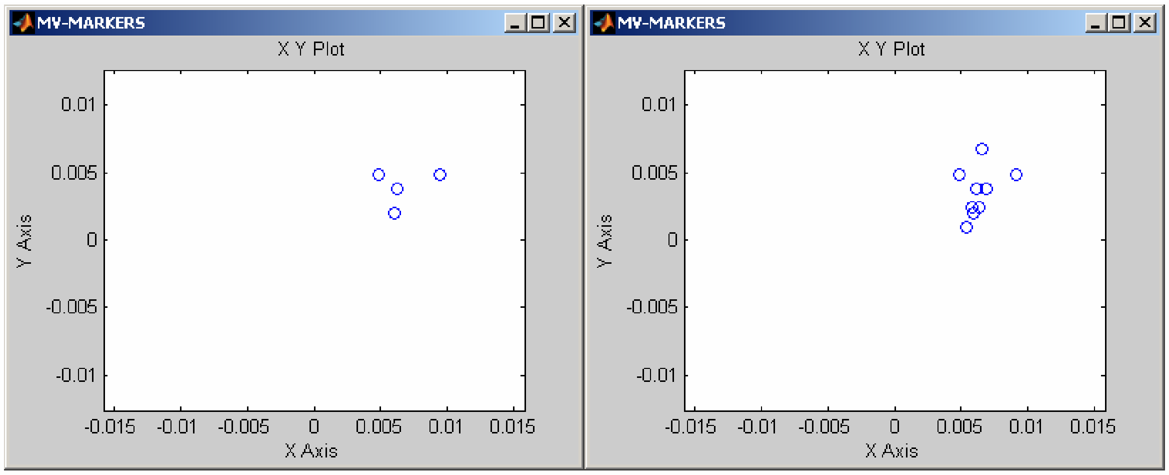


Fig. 4.10: RV (left) and SV (right) detected markers at the time t = 2.1 sec

4.2.6 The Pose estimation algorithm

The pose estimation algorithm block is the core of MV system, this block has one input that is the ordered vector of the coordinates $[u_1, \ v_1, \ \ldots\ldots, \ u_n, \ v_n]$ of the markers in the 2D CRF, which has been ordered by the labeling algorithm. If the k^{th} markers is not detectable the overflow value 100 is placed in the position *2* k* and *2*k+1*. The analyzed algorithms are two, the Gaussian Least Squares Differential Correction (GLSDC) [3] [27] and the Lu, Hager and Mjolsness algorithm (LHM) [4]. The GLSDC resolves the pose estimation problem finding the first order approximation for $[u_k, \ v_k], \ k = 1,..,n$ as a function of the 3D coordinates of the markers in camera frame, and solving the least square problem for the search of the differential correction using the iterative Gauss-Newton method. Advantages of this algorithms are its simplicity and speed, and the fact that it can provide the estimation within a fixed number of cycles. The main disadvantages are instead related to the fact that the first order approximation sometimes is not accurate enough, and can lead the algorithm to divergence. This in turn implies that the GLSDC is not robust with respect to errors in the markers position or initial conditions (see chapter 7). The LHM algorithms instead calculates the pose estimation minimizing an error metric based on collinearity in object space, this algorithm is iterative and computes the orthogonal rotation matrix. The iterative calculus without a fixed number of steps can be computationally more intensive than the GLSDC, especially if the input data are not smooth or the airplane considerably changes its position between the precedent and the current cycle of integration (0.1 sec). The LHM provides, on the other hand, high robustness and proved global convergence. The analysis of these algorithms and their performance is provided in the next 3 chapters of this effort.

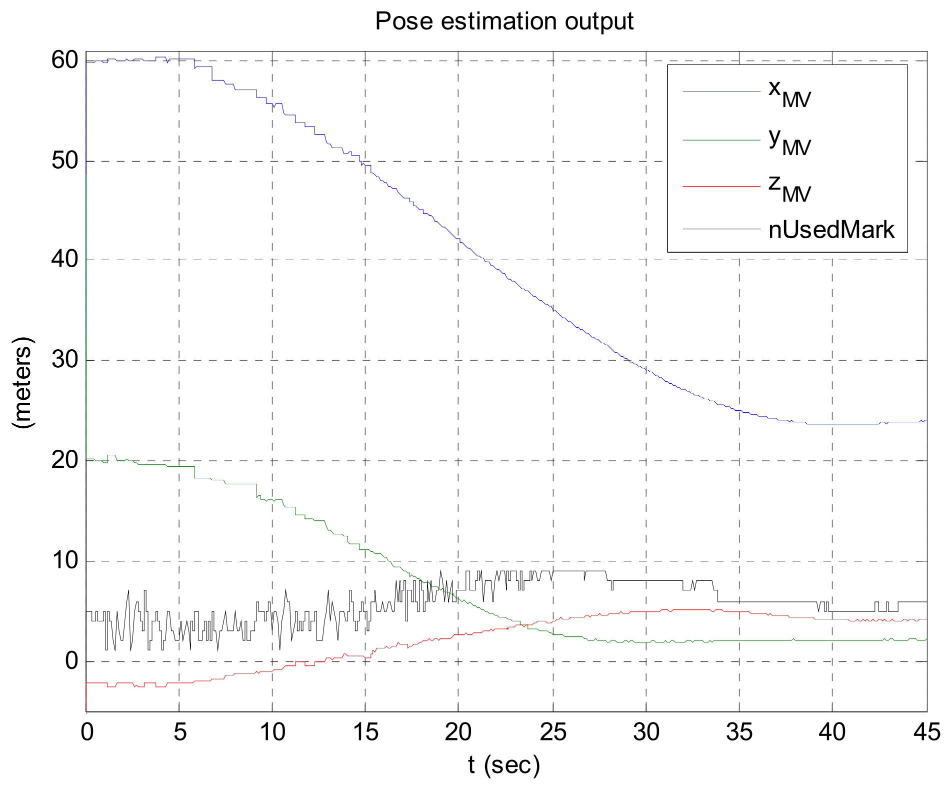


Fig. 4.11: LHM pose estimation output in RV case

The output of a pose estimation algorithm is the vector (*x, y, z, yaw, pitch, roll, nUsedMark*) that represents respectively the translation vector between TRF and CRF, the relative Euler angles between TRF and CRF and finally the number of markers used to provide the pose estimation. If we connect the first 6 parameter to the "S function3" (see Fig 4.2) we obtain an estimation of the homogeneous transformation matrix ${}^{C}T_{T}$.

5. THE GLSDC

Pose estimation is an essential step in many machine vision problems involving the estimation of an object's position and orientation relative to a model reference frame or relative to the object position and orientation at a previous time using a camera sensor or a range sensor. There are four pose estimation problems with point data. Each arises from two views taken of the same object that can be thought of as having undergone an unknown rigid body motion from the first view to the second view. In model-based vision, one "view" provides three-dimensional (3-D) data relative to the model reference frame. The other is the 2-D perspective projection. In motion estimation and structure from motion problems there is a rigid body motion of the sensor, the object or both. Both views are 2-D perspective projections. In any case, in each problem corresponding point pairs from the two views are obtained from some kind of matching procedure. The pose estimation problem with corresponding point data begins with such a corresponding point data set. Its solution is a procedure that uses the corresponding point data set to estimate the translation and rotation that define the relationship between the two coordinate frames.

In the simplest pose estimation problem, the data sets consist of two-dimensional data points in a two-dimensional space. Such data sets arise naturally when flat 3-D objects are viewed under perspective projection with the look angle being the same as the surface normal of the object viewed. In the next more difficult pose estimation problem, the data sets consist of three-dimensional data points in a three-dimensional space. Such data sets arise naturally when 3-D objects are viewed with a range finder sensor. In the most difficult pose estimation problems, one data set consists of the 2-D perspective projection of 3-D points and the other data set consists of either a 3-D point data set, in which case it is known as absolute orientation problem, or the other data set consists of a second 2-D perspective projection view of the same 3-D point data set, in which case, it is known as the relative orientation problem. The latter case occurs with time-varying imagery, uncontrolled stereo or multicamera imagery.

We are interested to the 2-D perspective projection of 3-D points problem but for the solution we need to know the solution of the 3-D – 3-D estimation problem.

5.1 3-D – 3-D estimation

Let $y_1,\ldots,y_N$ be N points in Euclidean 3-space. Let R be a rotation matrix and t be a translation vector. Let $x_1,\ldots x_N$ be the points in Euclidean 3-space that match $y_1,\ldots,y_N$, each x_N is the same rigid body motion of y_N. Hence each y_N is obtained as a rotation of x_N, plus a translation plus noise.

$$y_n = Rx_n + t + \eta_n \tag{5.1}$$

The 3-D-3-D pose estimation problem is to infer R and t from $x_1,\ldots x_N$ and $y_1,\ldots,y_N$.
To determine R and t we set up a constrained least-squares problem. We will minimize

$$\sum_{n=1}^{N} w_n \left\| y_n - (Rx_n + t) \right\|^2 \tag{5.2}$$

subject to the constraint that R is a rotation matrix, that is, $R^t=R^{-1}$. To be able to express these constraints using Lagrangian multipliers we let

$$R = \begin{pmatrix} r_1^t \\ r_2^t \\ r_3^t \end{pmatrix} \tag{5.3}$$

where each r_i is a 3x1 vector.
The constraint $R^t=R^{-1}$, then amounts to the six constant equation

$$
\begin{aligned}
r_1^t r_1 &= 1 \\
r_2^t r_2 &= 1 \\
r_3^t r_3 &= 1 \\
r_1^t r_2 &= 0 \\
r_1^t r_3 &= 0 \\
r_2^t r_3 &= 0
\end{aligned}
\qquad (5.4)
$$

The least-squares problem with constraints given by (5.4) can be written as minimizing ε^2 where

$$
\varepsilon^2 = \sum_{n=1}^{N}\sum_{k=1}^{3} w_n\left(y_{nk} - r_k^t x_n - t_k\right)^2 + \sum_{k=1}^{3}\lambda_k\left(r_k^t r_k - 1\right) + 2\lambda_4 r_1^t r_2 + 2\lambda_5 r_1^t r_3 + 2\lambda_6 r_2^t r_3
$$

$$
x_n = \begin{pmatrix} x_{n1} \\ x_{n2} \\ x_{n3} \end{pmatrix} \qquad y_n = \begin{pmatrix} y_{n1} \\ y_{n2} \\ y_{n3} \end{pmatrix} \qquad t_n = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} \qquad (5.5)
$$

Taking the partial derivate of ε^2 with respect to t_n, there results

$$
\frac{\partial \varepsilon^2}{\partial t_k} = \sum_{n=1}^{N} 2w_n\left(y_{nk} - r_k^t x_n - t_k\right)(-1) \qquad k = 1, 2, 3. \qquad (5.6)
$$

Setting these partial to zero results in

$$
\sum_{n=1}^{N} w_n\left(y_n - Rx_n - t\right) = 0 \qquad (5.7)
$$

By rearranging we obtain

$$
t = \overline{y} - R\overline{x} \qquad (5.8)
$$

where

$$
\overline{x} = \frac{\sum_{n=1}^{N} w_n x_x}{\sum_{n=1}^{N} w_n} \qquad \overline{y} = \frac{\sum_{n=1}^{N} w_n y_x}{\sum_{n=1}^{N} w_n} \qquad (5.9)
$$

Thus once R is known, t is quickly determined from (5.8). Substituting $\overline{x} - R\overline{y}$ for t in the definition of ε^2, there results

$$
\varepsilon^2 = \sum_{n=1}^{N} w_n \sum_{k=1}^{3}\left(y_{nk} - \overline{y} - r_k^t (x_n - \overline{x})\right)^2 + \sum_{k=1}^{3}\lambda_k\left(r_k^t r_k - 1\right) + 2\lambda_4 r_1^t r_2 + 2\lambda_5 r_1^t r_3 + 2\lambda_6 r_2^t r_3 \qquad (5.10)
$$

where

$$
\overline{x} = \begin{pmatrix} \overline{x}_1 \\ \overline{x}_2 \\ \overline{x}_3 \end{pmatrix} \qquad \overline{y} = \begin{pmatrix} \overline{y}_1 \\ \overline{y}_2 \\ \overline{y}_3 \end{pmatrix} \qquad (5.11)
$$

Now we take partial derivatives of ε^2 with respect to the components of each y_n. To write things more compactly, by $\partial \varepsilon^2 / \partial r_n$ we mean a 3 x *1* vector whose components are the partial derivatives of ε^2 with respect to each of the components of r_n. Then

$$\frac{\partial \varepsilon^2}{\partial r_1} = \sum_{n=1}^{N} 2w_n \left(y_{n1} - \bar{y}_1 - r_1^t (x_n - \bar{x})\right)\left(x_n - \bar{x}\right)(-1) + 2\lambda_1 r_1 + 2\lambda_4 r_2 + 2\lambda_5 r_3 \tag{5.12}$$

$$\frac{\partial \varepsilon^2}{\partial r_2} = \sum_{n=1}^{N} 2w_n \left(y_{n2} - \bar{y}_2 - r_2^t (x_n - \bar{x})\right)\left(x_n - \bar{x}\right)(-1) + 2\lambda_2 r_2 + 2\lambda_4 r_1 + 2\lambda_6 r_3 \tag{5.13}$$

$$\frac{\partial \varepsilon^2}{\partial r_3} = \sum_{n=1}^{N} 2w_n \left(y_{n3} - \bar{y}_3 - r_3^t (x_n - \bar{x})\right)\left(x_n - \bar{x}\right)(-1) + 2\lambda_3 r_3 + 2\lambda_5 r_1 + 2\lambda_6 r_2 \tag{5.14}$$

Setting these partial derivatives to zero and rearranging we obtain

$$\sum_{n=1}^{N} w_n \left(x_n - \bar{x}\right)\left(x_n - \bar{x}\right)^t r_1 + \lambda_1 r_1 + \lambda_4 r_2 + \lambda_5 r_3 = \sum_{n=1}^{N} w_n \left(y_{n1} - \bar{y}_1\right)\left(x_n - \bar{x}\right) \tag{5.15}$$

$$\sum_{n=1}^{N} w_n \left(x_n - \bar{x}\right)\left(x_n - \bar{x}\right)^t r_2 + \lambda_4 r_1 + \lambda_2 r_2 + \lambda_6 r_3 = \sum_{n=1}^{N} w_n \left(y_{n2} - \bar{y}_2\right)\left(x_n - \bar{x}\right) \tag{5.16}$$

$$\sum_{n=1}^{N} w_n \left(x_n - \bar{x}\right)\left(x_n - \bar{x}\right)^t r_3 + \lambda_5 r_1 + \lambda_6 r_2 + \lambda_3 r_3 = \sum_{n=1}^{N} w_n \left(y_{n3} - \bar{y}_3\right)\left(x_n - \bar{x}\right) \tag{5.17}$$

Let

$$A = \sum_{n=1}^{N} \left(x_n - \bar{x}\right)\left(x_n - \bar{x}\right)^t$$

$$\Lambda = \begin{bmatrix} \lambda_1 & \lambda_4 & \lambda_5 \\ \lambda_4 & \lambda_2 & \lambda_6 \\ \lambda_5 & \lambda_6 & \lambda_3 \end{bmatrix} \tag{5.18}$$

and

$$B = \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} \tag{5.19}$$

where

$$b_k = \sum_{n=1}^{N} w_n \left(y_{nk} - \bar{y}_k\right)\left(x_n - \bar{x}\right) \tag{5.20}$$

Then (5.15), (5.16) and (5.17) can be simply rewritten as

$$AR^t + R^t \Lambda = B \tag{5.21}$$

Multiplying both sides of (5.21) on the left by *R* we have

$$RAR^t + \Lambda = RB \tag{5.22}$$

Since $A=A^t$, $(RAR^t)^t=RAR^t$. Since both RAR^t and Λ are symmetric, the left-hand side must be symmetric. Hence the right-side is also symmetric. This means

$$RB = \left(RB\right)^t \tag{5.23}$$

The solution for *R* now comes quickly. Let the singular value decomposition of *B* be

$$B = UDV \tag{5.24}$$

where *U* and *V* are orthonormal and *D* is diagonal. Then

$$RUDV = \left(UDV\right)^t R^t = V^t D U^t R^t \tag{5.25}$$

By observation, a solution for R is immediately obtained as

$$R = V^t U^t \tag{5.26}$$

Solutions to this problem can be found in the photogrammetry literature beginning with Thompson [17], Schut [18], Tienstra [19], and Pope [20] . Blais [21] gives a solution to the problem in the case where there may be a scale factor or magnification different than 1. Sansò [22] gives a solution to the problem using quaternions. Arun et al. [23] and Haralick et al. [24] have discussed the singular value decomposition approach to the problem.

5.2 2-D perspective projection – 3-D pose estimation

Let $y_1, \ldots\ldots, y_N$ be the observed 3-D model points in Euclidean 3-space. Let R be a rotation matrix and t be a translation vector. Let (u_{n1}, u_{n2}), n =1,..., N be the corresponding 2-D perspective projection of the 3-D points. Then the relationship between the 3-D model points and the 2-D perspective projection points is given by

$$\begin{aligned} u_{n1} &= f\frac{r_1 y_n + t_1}{r_3 y_n + t_3} \\ u_{n2} &= f\frac{r_2 y_n + t_2}{r_3 y_n + t_3} \\ t &= (t_1, t_2, t_3)^t \\ R &= \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix} \end{aligned} \tag{5.27}$$

where f , the focal length, is the distance of the image plane in front of the origin that is the center of perspectivity. In the 3-D coordinate system of the camera, the perspective projections are given by

$$u_n = \begin{pmatrix} u_{n1} \\ u_{n2} \\ f \end{pmatrix} = f\begin{pmatrix} v_{n1} \\ v_{n2} \\ 1 \end{pmatrix} = f v_n \tag{5.28}$$

where $u_{n1} = f v_{n1}$ and $u_{n2} = f v_{n2}$.

The problem of pose estimation is to determine the unknown rotation matrix R and the translation vector t given the 3-D model points and the corresponding 2-D perspective projection points on the image plane. This problem is known as the exterior orientation problem in the photogrammetry literature. The dissertation by Szczepanski [14] surveys nearly 80 different solutions beginning with one given by Schrieber of Karlsruhe in the year 1879. The first robust solution in the computer vision literature was Fischler and Bolles [15]. Wrobel and Klemm [16] discuss the fact that there are configurations of points for which the solution is unstable.

5.2.1 Iterative least-square solution

This section describes iterative procedures for determining a least-squares solution for R and t **.** In the following subsections we use the superscript or subscript k to denote the values in the kth iteration step. Let

$$x_n = \begin{pmatrix} x_{n1} \\ x_{n2} \\ x_{n3} \end{pmatrix} = R\begin{pmatrix} y_{n1} \\ y_{n2} \\ y_{n3} \end{pmatrix} + t \tag{5.29}$$

be the rotated and translated point of y_n. Let d_n be the estimated depth of each point x_n relative to the camera coordinate system.

- **Method 1:** One iterative procedure for determining a least-square solution for *R* and *t* is the following.
    1. Choose initial reasonable values for the depth d_n^0 of each point. The initial values could, for example, be the same constant for each point, the constant representing an initial guess of how far the object is from the perspective center.
    2. Iterate. Suppose the depth values d_n^k, $n = 1,...,N$ are given. Define the depth values for the $(k$ +1)th iteration by:
        a) Find the rotation matrix R_k and the translation vector t_k that minimizes

        $$\varepsilon_k^2 = \sum_{n=1}^{N} w_n \left\| R_k y_n + t_k - d_n^k v_n \right\|^2 \tag{5.30}$$

        where the $\{w_n \mid n = 1,\dots,N\}$ are nonnegative weights reflecting the goodness of the observations. R_k and t_k constitute the solution to the 3-D-3-D pose estimation problem.
        b) Define

        $$d_n^{k+1} = \left(\frac{D_y}{D_x} \right) x_{n3}^k \tag{5.31}$$

        where

        $$\bar{x} = \frac{1}{n} \sum_{n=1}^{N} x_n, \qquad \bar{y} = \frac{1}{n} \sum_{n=1}^{N} y_n \tag{5.32}$$

        and

        $$D_y = \sum_{n=1}^{N} \left\| y_n - \bar{y} \right\|^2 \tag{5.33}$$

        $$D_x = \sum_{n=1}^{N} \left\| x_n - \bar{x} \right\|^2 \tag{5.34}$$

- **Method 2:** Replace the step 2b) of method 1 with step 1 of method 2.
    1. Define

    $$d_n^{k+1} = \frac{(R_k y_n + t_k)^t v_n}{v_n^t v_n} \tag{5.35}$$

It can be shown that $\varepsilon_{k+1}^2 \leq \varepsilon_k^2$ and

$$\begin{aligned}
\varepsilon_{k+1}^2 &= \sum_{n=1}^{N} w_n \left\| R_{k+1} y_n + t_{k+1} - d_n^{k+1} v_n \right\|^2 \le \sum_{n=1}^{N} w_n \left\| R_k y_n + t_k - d_n^{k+1} v_n \right\|^2 = \\
&= \sum_{n=1}^{N} w_n \left\| \left(x_n^k - d_n^k v_n \right) + \left(d_n^k v_n - d_n^{k+1} v_n \right) \right\|^2 = \\
&= \sum_{n=1}^{N} w_n \left[\left\| x_n^k - d_n^k v_n \right\|^2 + 2 \left(x_n^k - d_n^k v_n \right)^t \left(d_n^k - d_n^{k+1} \right) v_n + \left(d_n^k - d_n^{k+1} \right) \left\| v_n \right\|^2 \right] = \\
&= \varepsilon_k^2 + \sum_{n=1}^{N} w_n \left(d_n^k - d_n^{k+1} \right) \left[2 x_n^{k\,t} v_n - 2 d_n^k \left\| v_n \right\|^2 + \left(d_n^k - d_n^{k+1} \right) \left\| v_n \right\|^2 \right] = \\
&= \varepsilon_k^2 + \sum_{n=1}^{N} w_n \left(d_n^k - d_n^{k+1} \right) \left[2 x_n^{k\,t} v_n - \left(d_n^k - d_n^{k+1} \right) \left\| v_n \right\|^2 \right] = \\
&= \varepsilon_k^2 + \sum_{n=1}^{N} w_n \left\| v_n \right\|^2 \left[\left(d_n^{k+1} \right)^2 - 2 \frac{x_n^{k\,t} v_n}{\left\| v_n \right\|^2} d_n^{k+1} + 2 \frac{x_n^{k\,t} v_n}{\left\| v_n \right\|^2} d_n^k - \left(d_n^k \right)^2 \right] = \\
&= \varepsilon_k^2 + \sum_{n=1}^{N} w_n \left\| v_n \right\|^2 \left[\left(d_n^{k+1} - \frac{x_n^{k\,t} v_n}{\left\| v_n \right\|^2} \right)^2 - \left(d_n^k - \frac{x_n^{k\,t} v_n}{\left\| v_n \right\|^2} \right)^2 \right].
\end{aligned} \tag{5.36}$$

Consider the terms in the bracket as a function of d_n^{k+1}. The function reaches a minimum when

$$d_n^{k+1} = \frac{x_n^{k\,t} v_n}{\left\| v_n \right\|^2} \tag{5.37}$$

The resulting value of the terms in the bracket at the minimum is

$$- \left(d_n^k - \frac{x_n^{k\,t} v_n}{\left\| v_n \right\|^2} \right)^2 \tag{5.38}$$

This value cannot be positive. Since $w_n \left\| v_n \right\|^2 > 0$, when

$$d_n^{k+1} = \frac{x_n^{k\,t} v_n}{\left\| v_n \right\|^2} \tag{5.39}$$

Each term in the summation is not positive and from this we can infer

$$\varepsilon_{k+1}^2 \le \varepsilon_k^2 \tag{5.40}$$

5.2.2 Least-squares adjustment by linearization

Let ϕ, θ, ψ be the three angles that define the rotation matrix R such that

$$R = R_x(\phi) R_y(\theta) R_z(\psi) = \\
= \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & \sin\phi\cos\theta \\ \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi & \cos\phi\cos\theta \end{bmatrix} \tag{5.41}$$

As there always exists random errors in the measurement of the image coordinates, let

$$u_{ni} = u_{ni}^0 + v_{ni}, \quad i = 1,2. \quad n = 1,....,N \tag{5.42}$$

where $\left(u_{n1}^0, u_{n2}^0\right)$ are the measured image points and $\left(v_{n1}, v_{n2}\right)$ are the correction needed to account for the random error in the measured coordinates. Similarly, let

$$\begin{aligned} \phi &= \phi^0 + \Delta\phi \\ \theta &= \theta^0 + \Delta\theta \\ \psi &= \psi^0 + \Delta\psi \\ t_i &= t_i^0 + \Delta t_i, \quad i = 1, 2, 3 \end{aligned} \tag{5.43}$$

where $\phi^0, \theta^0, \psi^0, t_1^0, t_2^0, t_3^0$ are some approximation, and $\Delta\phi, \Delta\theta, \Delta\psi, \Delta t_1, \Delta t_2, \Delta t_3$ are their corresponding correction. We assume that the correction Δ's are small and the collinearity equations are linear over the small intervals between the true values of these parameters and their corresponding approximation.

Let

$$\begin{aligned} F_{n1} &= u_{n1} - f\frac{r_1 y_n + t_1}{r_3 y_n + t_3} \\ F_{n2} &= u_{n2} - f\frac{r_2 y_n + t_2}{r_3 y_n + t_3} \end{aligned} \tag{5.44}$$

These equation can be linearized by Newton's first order approximation as follows:

$$\begin{aligned} F_{n1} &\cong F_{n1}^0 + v_{n1} + b_{n11}\Delta\phi + b_{n12}\Delta\theta + b_{n13}\Delta\psi + b_{n14}\Delta t_1 + b_{n15}\Delta t_2 + b_{n16}\Delta t_3 \\ F_{n2} &\cong F_{n2}^0 + v_{n2} + b_{n21}\Delta\phi + b_{n22}\Delta\theta + b_{n23}\Delta\psi + b_{n24}\Delta t_1 + b_{n25}\Delta t_2 + b_{n26}\Delta t_3 \end{aligned} \tag{5.45}$$

where

$$\begin{aligned} b_{ni1} &= \left(\frac{\partial F_{ni}}{\partial\phi}\right)^0 & b_{ni2} &= \left(\frac{\partial F_{ni}}{\partial\theta}\right)^0 \\ b_{ni3} &= \left(\frac{\partial F_{ni}}{\partial\psi}\right)^0 & b_{ni4} &= \left(\frac{\partial F_{ni}}{\partial t_1}\right)^0 \\ b_{ni5} &= \left(\frac{\partial F_{ni}}{\partial t_3}\right)^0 & b_{ni6} &= \left(\frac{\partial F_{ni}}{\partial t_3}\right)^0 \end{aligned} \tag{5.46}$$

for *i* = 1,2, where the superscript 0 implies that the function values are computed with the approximations ($\phi^0, \theta^0, \psi^0, t_1^0, t_2^0, t_3^0$). Taking F_{nl} = F_{n2} = 0, the linearized equation can be expressed as the matrix system

$$\begin{bmatrix} b_{111} & b_{112} & b_{113} & b_{114} & b_{115} & b_{116} \\ b_{121} & b_{122} & b_{123} & b_{124} & b_{125} & b_{126} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ b_{N11} & b_{N12} & b_{N13} & b_{N14} & b_{N15} & b_{N16} \\ b_{N21} & b_{N22} & b_{N23} & b_{N24} & b_{N25} & b_{N26} \end{bmatrix} \begin{bmatrix} \Delta\phi \\ \Delta\theta \\ \Delta\psi \\ \Delta t_1 \\ \Delta t_2 \\ \Delta t_3 \end{bmatrix} = \begin{bmatrix} -F_{11}^0 \\ -F_{12}^0 \\ \dots. \\ -F_{N1}^0 \\ -F_{N2}^0 \end{bmatrix} - \begin{bmatrix} v_{11} \\ v_{12} \\ \dots. \\ v_{N1} \\ v_{N2} \end{bmatrix} \tag{5.47}$$

or simply

$$B\Delta = F - v \tag{5.48}$$

This equation can be solved using the singular value decomposition method. The computed corrections $\Delta = \left(\Delta\phi, \Delta\theta, \Delta\psi, \Delta t_1, \Delta t_2, \Delta t_3\right)^t$ from one iteration are used to update the parameters

$\Lambda = \left(\phi^0, \theta^0, \psi^0, t_1^0, t_2^0, t_3^0\right)^t$ and then these updated parameters are used as approximations in the next iteration. The whole iteration process is repeated until the corrections become negligibly small.

5.3 The GLSDC implementation

The evaluation of the rigid transformation from camera to tanker, is a typical pose estimation problem [3]. In this effort, the information available for solving the problem is given in the form of a set of point correspondences each composed of a 3-D reference point (marker) expressed in object coordinates (Tanker Reference Frame TRF) and its 2-D projection expressed in image (u,v) coordinates (Camera Reference Frame CRF). For an arbitrary number of points, this approach to pose estimation is based upon the application the Gauss-Newton method [3], [25], [26], [4] to the minimization of a nonlinear cost function typically solved iteratively using. The Gaussian Least Squares Differential Correction (GLSDC) algorithm has been implemented in [27]. This algorithm exhibited convergence and accuracy even in presence of quantization noise produced by the CCD matrix. The nonlinear 3-D to 2-D correspondence in terms of the unknown vector $X = [{}^C x_t, {}^C y_t, {}^C z_t, {}^C \psi_t, {}^C \theta_t, {}^C \varphi_t]^t$ and known vectors ${}^T P_{(j)}$, where ${}^T P_{(j)}$ are the marker positions in TRF, can be written as:

$$\begin{bmatrix} g_{uj} \\ g_{vj} \end{bmatrix} = \begin{bmatrix} g_u\left(f, X, {}^T P_{(j)}\right) \\ g_v\left(f, X, {}^T P_{(j)}\right) \end{bmatrix} \tag{5.49}$$

with $j = 1, \dots, m$. By grouping the equation (5.49) for all *m* markers, the following *1* x *2m* vector of nonlinear relationships is generated

$$G = \left[g_{u1}, g_{v1}, \dots, g_{um}, g_{vm}\right] \tag{5.50}$$

Next, the MV estimation error at sampling time *k* is defined as:

$$\Delta G(k) = G_{meas}(k) - G\left(f, \hat{X}(k), {}^T P_{(j)}(k)\right) \tag{5.51}$$

where $G_{meas}(k)$ is a vector of the measured coordinates of the markers on the image plane and $\hat{X}(k)$ is the current estimation of vector *X*. In each time frame the starting estimation of *X* is iteratively refined by the GLSDC algorithm by repeating the following steps for a number of iterations (with index *i*):

$$R_i(k) = A_i^t(k) W_k A_i(k) \tag{5.52}$$

$$\Delta\hat{X}_i(k) = R_i^{-1}(k) A_i^t(k) W_k \Delta G_i(k) \tag{5.53}$$

$$\hat{X}_{i+1}(k) = \hat{X}_i(k) + \Delta\hat{X}_i(k) \tag{5.54}$$

In (5.52) the matrix *A* is a *2m* x *6* Jacobian matrix

$$A_i(k) = \left.\frac{\partial G_i(k)}{\partial X(k)}\right|_{X = \hat{X}_i(k)} \tag{5.55}$$

and *W* is the 2*m* x 2*m* covariance matrix $W = diag\left(1/\sigma_{u1}, 1/\sigma_{v1}, \dots, 1/\sigma_{um}, 1/\sigma_{vm}\right)$ of the estimation error. The initial guess $\hat{X}_0(k)$ at time step *k* is given by the final estimation at time step *k-1*.

The basic algorithm (5.52)-(5.55) is designed to work with a fixed number of *m* markers. The following strategy has been introduced for handling a time varying number of markers. At the beginning of each time step the number of the visible markers is evaluated; in the event that some markers are not visible, these are removed by the estimation process. This entails that (5.50) has to be modified with the appropriate number of rows; next, the dimensions and the values of the matrices *A* and *W* in (5.51) - (5.55) are adjusted accordingly. The internal function that provides the pose estimation (5.52)-(5.55) is called a fixed numbers of times that in a *nominal case* is 3. The algorithm is able to estimate the position with a minimum number of detected markers of 4. The simulations associated with the described modification of the GLSDC algorithm will be shown in the analysis described in chapter 7.

6. THE LHM

The LHM algorithm formulates the pose estimation problem as that of minimizing an *object-space* collinearity error. From this objective function, we derive an algorithm that operates by successively improving an estimate of the rotation portion of the pose and then estimates an associated translation. The intermediate rotation estimates are always the best orthogonal solution for each iteration. The orthogonality constraint is enforced by using singular value decomposition, not from specific parameterization of rotations, e.g., Euler angles. We further prove that the proposed algorithm is globally convergent.

6.1 Camera model

The mapping from 3D reference points to 2D image coordinates can be formalized as follows: Given a set of noncollinear 3D coordinates of reference points $p_i = (x_i, y_i, z_i)^t$, i =1,..,n, n ≥ 3 expressed in an object-entered reference frame, the corresponding camera-space coordinates $q_i = (x'_i, y'_i, z'_i)^t$, are related by a rigid transformation as:

$$q_i = Rp_i + t \tag{6.1}$$

where

$$R = \begin{pmatrix} r_1^t \\ r_2^t \\ r_3^t \end{pmatrix} \in SO(3) \quad \text{and} \quad t = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \in \Re^3 \tag{6.2}$$

are a rotation matrix and a translation vector, respectively. The camera reference frame is chosen so that the center of projection of the camera is at the origin and the optical axis points in the positive *z* direction. The reference points p_i are projected to the plane with $z' = 1$, referred to as the *normalized image plane*, in the camera reference frame. Let the image point $\mathbf{v}_i = (u_i, v_i, 1)^t$ be the projection of p_i on the normalized image plane (see Fig 6.1). Under the idealized pinhole imaging model, $\mathbf{v}_i$, q_i and the center of projection are collinear. This fact is expressed by the following equation:

$$u_i = \frac{r_1^t p_i + t_x}{r_3^t p_i + t_z} \tag{6.3a}$$

$$v_i = \frac{r_2^t p_i + t_y}{r_3^t p_i + t_z} \tag{6.3b}$$

or

$$\mathbf{v}_i = \frac{1}{r_3^t p_i + t_z}(Rp_i + t) \tag{6.4}$$

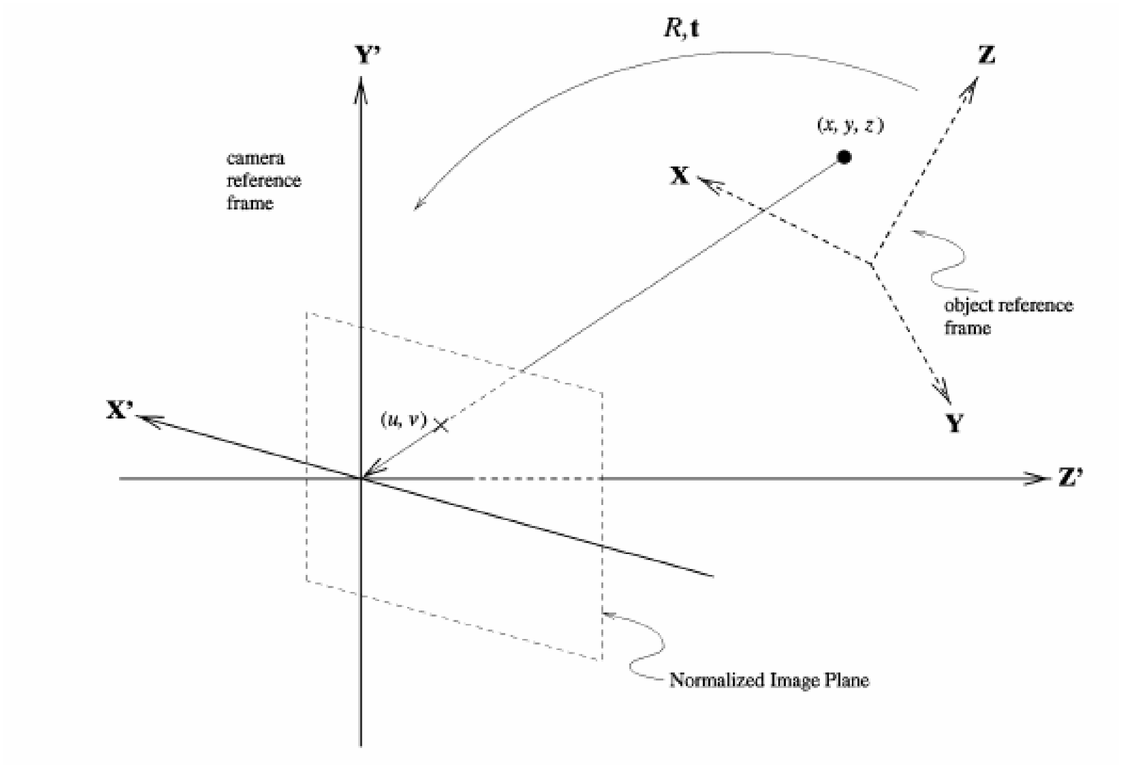


Fig. 6.1: the reference frame in pose estimation problem

which is known as the *collinearity equation* in the photogrammetry literature. However, another way of thinking of collinearity is that the orthogonal projection of q_i on $\mathbf{v}_i$ should be equal to q_i itself. This fact is expressed by the following equation:

$$Rp_i + t = V_i(Rp_i + t) \tag{6.5}$$

where

$$V_i = \frac{\mathbf{v}_i\mathbf{v}_i^t}{\mathbf{v}_i^t\mathbf{v}_i} \tag{6.6}$$

is the line-of-sight projection matrix that, when applied to a scene point, projects the point orthogonally to the line of sight defined by the image point $\mathbf{v}_i$. Since V_i is a projection operator, it satisfies the following properties:

$$\|\mathbf{x}\| \geq \|V_i\mathbf{x}\|, \quad \mathbf{x} \in \Re^3, \tag{6.7a}$$

$$V_i^t = V_i \tag{6.7b}$$

$$V_i^2 = V_iV_i^t = V_i \tag{6.7c}$$

In the remainder of this chapter, we refer to (6.4) as the *image space* collinearity equation and (6.5) as the *object space* collinearity equation. The pose estimation problem is to develop an algorithm for finding the rigid transform *(R, t)* that minimizes some form of accumulation of the errors (for example, summation of squared errors) of either of the collinearity equations (see Fig. 6.2).

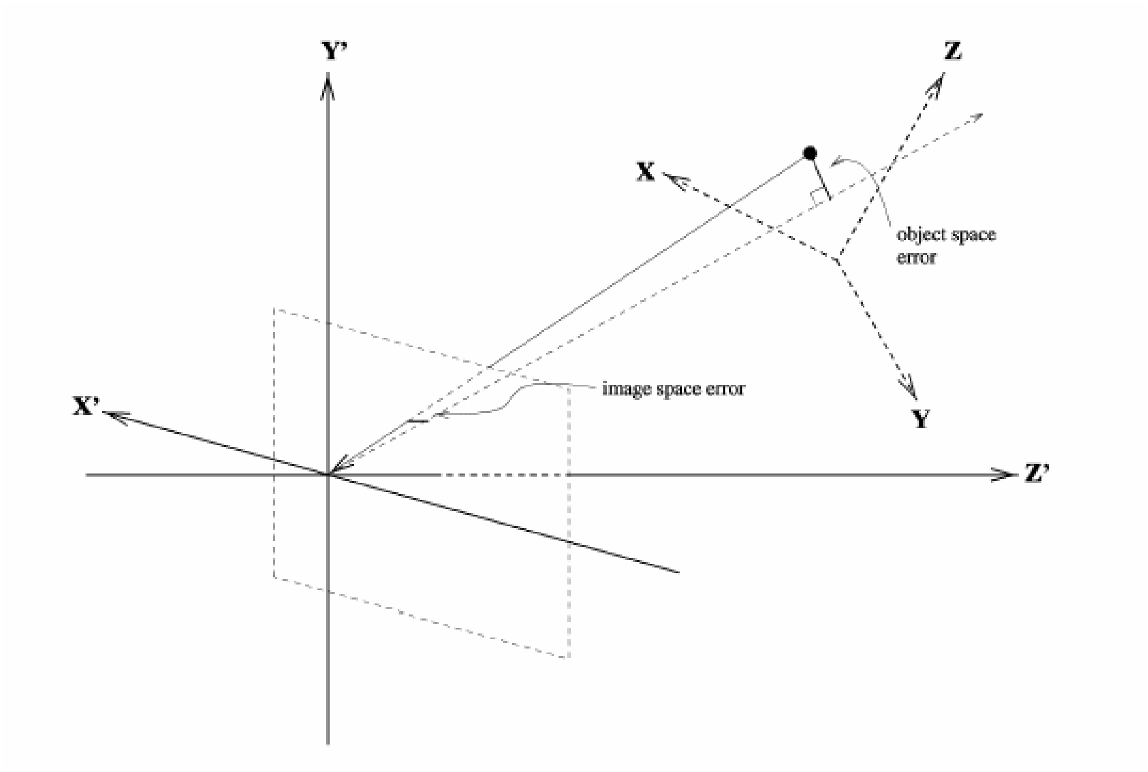


Fig. 6.2: object-space and image space collinearity errors

6.2 The LHM algorithm

The main objective of this section is to explain the LHM algorithm (also referred to as the *orthogonal iteration* (OI) method) in more detail, using the previously introduced formulation. In particular, the pose estimation problem is firstly introduced, using an appropriate object space error function, then this function is rewritten in a way which admits an iteration based on the solution to the 3D-3D pose estimation problem. Since the algorithm depends heavily on the solution to absolute orientation, we first review the absolute orientation problem and its solution before presenting the algorithm and proving its convergence.

6.2.1 Optimal absolute orientation solution

The absolute orientation problem can be posed as follows: suppose the 3D camera-space coordinates q_i could be reconstructed physically (for example, by range sensing) or computationally (for example, by stereo matching or structure-from-motion). Then, for each observed point, we have:

$$q_i = Rp_i + t \tag{6.8}$$

Computing absolute orientation is the process of determining R and t from corresponding pairs q_i and p_i. With three or more noncollinear reference points, R and t can be obtained as a solution to the following least-squares problem

$$\min_{R,t} \sum_{i=1}^{n} \left\| Rp_i + t - q_i \right\|^2, \quad \text{subject to } R^t R = I \tag{6.9}$$

Such a constrained least-squares problem [6] can be solved in closed form using quaternions [7], [8] or singular value decomposition (SVD) [12], [9], [7], [8].

The SVD solution proceeds as follows: Let $\{ p_i\}$ and $\{ q_i\}$ denote lists of corresponding vectors related by (6.1) and define

$$\bar{p} \overset{def}{=} \frac{1}{n}\sum_{i=1}^{n} p_i\,, \quad \bar{q} \overset{def}{=} \frac{1}{n}\sum_{i=1}^{n} q_i \tag{6.10}$$

that is, $\bar{p}$ and $\bar{q}$ are the centroid of $\{p_i\}$ and $\{q_i\}$, respectively. Define

$$p_i' = p_i - \bar{p}\,, \quad q_i' = q_i - \bar{q} \tag{6.11}$$

and

$$M = \sum_{i=1}^{n} q_i' p_i'^t \tag{6.12}$$

In other words, $\frac{1}{n}M$ is the sample cross-covariance matrix between $\{p_i\}$ and $\{q_i\}$. It can be shown that [12] if R^*, t^* minimize (6.9), then they satisfy

$$R^* = \arg\max_R tr(R^t M) \tag{6.13}$$

$$t^* = \bar{q} - R^* \bar{p} \tag{6.14}$$

Let (U, Σ, V) be a SVD of M, that is, $U^t M V = \Sigma$. Then, the solution to (6.9) is

$$R^* = VU^t \tag{6.15}$$

Note that the optimal translation is entirely determined by the optimal rotation and all information for finding the best rotation is contained in M as defined in (6.12). Hence, only the position of the 3D points relative to their centroids is relevant in the determination of the optimal rotation matrix. It is also important to note that, although the SVD of a matrix is not unique, the optimal rotation is the one shown in Appendix 10.2.

6.2.3 The algorithm

We now turn to the development of the LHM Algorithm. The starting point for the algorithm is to state the pose estimation problem using the following *object-space* collinearity error vector (see Fig. 6.2):

$$e_i = (I - \hat{V}_i)(Rp_i + t) \tag{6.16}$$

where $\hat{V}_i$ is the observed line-of-sight projection matrix defined as:

$$\hat{V}_i = \frac{\hat{\mathbf{v}}_i \hat{\mathbf{v}}_i^t}{\hat{\mathbf{v}}_i^t \hat{\mathbf{v}}_i} \tag{6.17}$$

We then seek to minimize the sum of the squared error

$$E(R,t) = \sum_{i=1}^{n} \|e_i\|^2 = \sum_{i=1}^{n} \left\|(I - \hat{V}_i)(Rp_i + t)\right\|^2 \tag{6.18}$$

over R and t. Note that all the information contained in the set of the observed image points $\{\mathbf{v}_i\}$ is now completely encoded in the set of projection matrices $\{\hat{V}_i\}$. Since this objective function is quadratic in t, given a fixed rotation R, the optimal value for t can be computed in closed form as:

$$t(R) = \frac{1}{n}\left(I - \frac{1}{n}\sum_j \hat{V}_j\right)^{-1} \sum_j (\hat{V}_j - I)Rp_j \tag{6.19}$$

for (6.19) to be well-defined, $I - \frac{1}{n}\sum_{i=1}^{n} \hat{V}_i$ must be positive definite, which can be verified as follows:

For any 3 - vector $x \in \Re^3$, it can shown that

$$x^t\left(I-\frac{1}{n}\sum_{i=1}^{n}\hat{V}_i\right)x=\frac{1}{n}\sum_{i=1}^{n}\left(\|x\|^2-x^t\hat{V}_i x\right)=\frac{1}{n}\sum_{i=1}^{n}\left(\|x\|^2-x^t\hat{V}_i^t\hat{V}_i x\right) \tag{6.20}$$

$$=\frac{1}{n}\sum_{i=1}^{n}\left(\|x\|^2-\left\|\hat{V}_i x\right\|^2\right)>0$$

While $\|x\|^2-\left\|\hat{V}_i x\right\|^2$ can be individually greater than or equal to zero, they cannot be all equal to zero unless all scene points are projected to the same image point. Therefore, (6.20) is generally strictly greater than zero and, thus, the positive definiteness of $\hat{V}_i$ is asserted.
Given the optimal translation as a function of *R* and defining

$$q_i(R)\overset{def}{=}\hat{V}_i(Rp_i+t(R)) \quad \text{and} \quad \overline{q}(R)\overset{def}{=}\frac{1}{n}\sum_{i=1}^{n}q_i(R) \tag{6.21}$$

(6.18) can be rewritten as:

$$E(R)=\sum_{i=1}^{n}\left\|(Rp_i+t(R)-q_i(R))\right\|^2 \tag{6.22}$$

This equation now bears a close resemblance to the absolute orientation problem (compare with (6.9)). Unfortunately, in this case, we cannot solve for *R* in closed form as the sample cross-covariance matrix between $\{p_i\}$ and $\{q_i(R)\}$, that is,

$$M(R)=\sum_{i=1}^{n}q_i'(R)p_i'^t \tag{6.23}$$

where $p_i'=p_i-\overline{p}$, and $q_i'(R)=q_i(R)-\overline{q}(R)$ is dependent on *R* itself.
However, *R* can be computed iteratively as follows: first assume that the *k*th estimate of *R* is $R^{(k)}$, $t^{(k)}=t(R^{(k)})$ and $q_i^{(k)}=R^{(k)}p_i+t^{(k)}$. The next estimate $R^{(k+1)}$, is determined by solving the following absolute orientation problem:

$$R^{(k+1)}=\arg\min_R\sum_{i=1}^{n}\left\|Rp_i+t-\hat{V}_i q_i^{(k)}\right\|^2= \tag{6.24}$$

$$=\arg\max_R tr\left(R^t M(R^{(k)})\right) \tag{6.25}$$

where the set of $\hat{V}_i q_i^{(k)}$ is treated as a hypothesis of the set of the scene points q_i in (6.9). In this form, the solution for $R^{(k+1)}$ is given by (6.15). We then compute the next estimate of translation, using (6.19), as:

$$t^{(k+1)}=t\left(R^{(k+1)}\right) \tag{6.26}$$

and repeat the process. A *solution* R^* to the pose estimation problem using the LHM algorithm is defined to be a fixed point to (6.24), that is, R^* satisfies

$$R^*=\arg\min_R\sum_{i=1}^{n}\left\|Rp_i+t-\hat{V}_i(R^*p_i+t(R^*))\right\|^2 \tag{6.27}$$

Note that is a *solution* does not necessary correspond to the *correct* true pose.

6.2.4 Global convergence

We now wish to show that the orthogonal iteration algorithm will converge to an optimum of (25) for any set of observed points and any starting point $R^{(0)}$. Our proof, which is based on the Global Convergence Theorem [13, chapter 6], requires the following definitions:

Def. 6.1: *A point-to-set mapping* **A** *from X to Y is said to be* closed *at* $x \in X$ *if the assumption*

1. $x_k \to x, \quad x_k \in X$
2. $y_k \to y, \quad y_k \in \mathbf{A}(x_k)$ imply
3. $y \in \mathbf{A}(x)$

The point-to-set mapping **A** *is said to be* closed *on X if it is closed at each point of X.*
Note the continuous point-to-point mapping are special closed point-to-set mappings.

Def. 6.2: *A set* **S** *is said to be closed if* $x_k \to x$ with $x_k \in \mathbf{S}$ implies $x \in \mathbf{S}$. **S** *is said to be* compact *if it is both closed and bounded.*

Define $\mathbf{OI} : SO(3) \to SO(3)$ to be the mapping that generates $R^{(k+1)}$ from $R^{(k)}$, that is, $R^{(k+1)}=\mathbf{OI}(R^{(k)})$. According to the Global Convergence Theorem [13], to prove the global convergence of the orthogonal iteration algorithm we need to show that

1. **OI** is closed.
2. All $\{ R^{(k)} \}$ generated by OI are contained in a compact set.
3. **OI** strictly decreases the objective function unless a solution is reached.

To verify the first condition, we note that **OI** can be considered as the composition of three mappings:

$F : SO(3) \to \Re^{3\times3}$ is a point-to-point mapping that represents the computation $M^{(k)} = M(R^{(k)})$ in (6.23).

$SVD : \Re^{3\times3} \to SO(3) \times \wp\ell(3) \times SO(3)$ is a point-to-set mapping that represents the calculation of the SVD of $M^{(k)}$.

$G : SO(3) \times \wp\ell(3) \times SO(3) \to SO(3)$ is a point-to-point mapping that represents the computation of $R^{(k+1)}$ from the SVD of $M(R^{(k)})$ using (6.15).

Where $SO(3)$ is the set 3 x 3 orthogonal matrices and $\wp\ell(3)$ is the set 3 x 3 diagonal matrices.

The first and the last mappings, F and G, are continuous and, hence, are closed. In Appendix, it is shown that *SVD* is also a closed mapping. Therefore, it follows that **OI** is closed using the fact that the composition of closed mappings is also closed [13].

Since **OI** always generates orthogonal matrices and the set of orthogonal matrices $SO(3)$ is compact (closed and bounded), the second criteria is met.

Finally, we seek to prove the third criteria. The sum of squared error of the estimate $R^{(k+1)}$ can be related to that of $R^{(k)}$ as follows:

$$\begin{aligned} R^{(k+1)} &= \sum_{i=1}^{n} \left\| q_i^{(k+1)} - \hat{V}_i q_i^{(k+1)} \right\|^2 = \sum_{i=1}^{n} \left\| q_i^{(k+1)} - \hat{V}_i q_i^{(k)} + \hat{V}_i q_i^{(k)} - \hat{V}_i q_i^{(k+1)} \right\|^2 = \\ &= \sum_{i=1}^{n} \left\| q_i^{(k+1)} - \hat{V}_i q_i^{(k)} \right\|^2 + \sum_{i=1}^{n} \left(q_i^{(k)} - q_i^{(k+1)} \right) \hat{V}_i^t \left(2\left(q_i^{(k+1)} - \hat{V}_i q_i^{(k)} \right) + \hat{V}_i q_i^{(k)} - \hat{V}_i q_i^{(k+1)} \right) \end{aligned} \tag{6.28}$$

Applying the fact that $\hat{V}_i = \hat{V}_i \hat{V}_i^t$ to the second term in the right hand side of the last equation in (6.28), we have

$$\sum_{i=1}^{n} \left(2\left(\hat{V}_i q_i^{(k)} \right)^t \hat{V}_i q_i^{(k+1)} - 2\left\| \hat{V}_i q_i^{(k+1)} \right\|^2 - \left\| \hat{V}_i q_i^{(k)} \right\|^2 + \left\| \hat{V}_i q_i^{(k+1)} \right\|^2 \right) = -\sum_{i=1}^{n} \left\| \hat{V}_i q_i^{(k+1)} - \hat{V}_i q_i^{(k)} \right\|^2 \tag{6.29}$$

But, according to (6.24) and (6.26),

$$\sum_{i=1}^{n} \left\| q_i^{(k+1)} - \hat{V}_i q_i^{(k)} \right\|^2 \le \sum_{i=1}^{n} \left\| q_i^{(k)} - \hat{V}_i q_i^{(k)} \right\|^2 = E\left(R^{(k)} \right) \tag{6.30}$$

and we obtain

$$E\left(R^{(k+1)}\right) \le E\left(R^{(k)}\right) - \sum_{i=1}^{n} \left\| \hat{V}_i q_i^{(k+1)} - \hat{V}_i q_i^{(k)} \right\|^2 . \tag{6.31}$$

Assume that $R^{(k)}$ is not a fixed point of **OI** which implies $R^{(k+1)} \neq R^{(k)}$ and $q_i^{(k+1)} \neq q_i^{(k)}$. If $\sum_{i=1}^{n} \left\| \hat{V}_i q_i^{(k+1)} - \hat{V}_i q_i^{(k)} \right\|^2$ is equal to zero, then $\hat{V}_i q_i^{(k+1)} = \hat{V}_i q_i^{(k)}$. But since the optimal solution to the absolution orientation problem is unique, according to (6.24), we must have $R^{(k+1)} = R^{(k)}$, which contradicts our assumption that $R^{(k)}$ is not a fixed point. Therefore, $\sum_{i=1}^{n} \left\| \hat{V}_i q_i^{(k+1)} - \hat{V}_i q_i^{(k)} \right\|^2$ cannot be zero. Combined with (6.31), we have

$$E\left(R^{(k+1)}\right) < E\left(R^{(k)}\right) \tag{6.32}$$

meaning that **OI** decreases E strictly unless a solution is reached.

Now, we can claim that the orthogonal iteration algorithm is globally convergent, that is, a solution, or a fixed point, will eventually be reached from arbitrary starting point. Although global convergence does not guarantee that the true pose will always be recovered, it does suggest that the true pose can be reached from very a broad range of initial guesses. Based on the experiments we have empirically observed that the only constraint on $R^{(0)}$ for **OI** to recover the true pose is that it does not result in translation with negative z component, i.e., it does not place the object behind the camera.

6.2.5 Initialization and Weak Perspective approximation

The **OI** algorithm can be initiated as follows: Given an initial guess $R^{(0)}$ of R, compute $t^{(0)}$. The initial pose ($R^{(0)}$, $t^{(0)}$) is then used to establish a set of hypothesized scene points $\hat{V}_i(R^{(0)} p_i + t^{(0)})$, which are used to start the first absolute orientation iteration. Although the orthogonal iteration algorithm is globally convergent, it does not guarantee that it will efficiently or eventually converge to the correct solution. Instead of choosing $R^{(0)}$, we can treat vi themselves as the first hypothesized scene points. This leads to an absolute orientation problem between the set of 3D reference points p_i and the set of image points $\mathbf{v}_\mathrm{i}$ considered as coplanar 3D points. This initial absolute orientation problem is related to weak perspective approximation.

6.2.5.1 Weak-perspective model

Weak-perspective is an approximation to the perspective camera model described in Section 6.1. Under the weak perspective model, we have the following relation for each reference point p_i

$$u_i \approx \frac{1}{s}\left(r_1^t p_i + t_x\right) \tag{6.33a}$$

$$v_i \approx \frac{1}{s}\left(r_2^t p_i + t_y\right) \tag{6.33b}$$

where s is called *scale* or *principle depth*. Weak perspective is valid when the depths of all camera-space coordinates are roughly equal to the principle depth and the object is close to the optical axis of the camera. Conventionally, the principle depth is chosen as the depth of the origin of the object space, that is, the z -component of the translation t_z when $\bar{p}$, the center of the reference points, is also the origin of the object space. Here, we decouple the scale s from t_z, so it can be chosen as the one that minimizes its deviation from the depths of the camera space coordinates

$$\sum_{i=1}^{n} (r_3^t p_i + t_z - s)^2 \tag{6.34}$$

Of course, we also need to minimize the square of the image error over R, t and s

$$\sum_{i=1}^{n}\left[(r_1^t p_i + t_x - s\hat{u}_i)^2 + (r_2^t p_i + t_y - s\hat{v}_i)^2\right] \tag{6.35}$$

Combining (6.34) and (6.35), and weighting them equally, we have the following least-squares objective function:

$$\sum_{i=1}^{n}\left\| Rp_i + t - s\hat{\mathbf{v}}_i \right\|^2 \tag{6.36}$$

This is the same objective function as for absolute orientation, (6.9), but with the additional scale variable and the (implicit) constraint that all camera-space coordinates have the same depth. In this new objective function, the value of *s* together with *R* and *t* must be determined simultaneously.
By considering the following modified objective function [7], [12]

$$\min_{R,t,s}\sum_{i=1}^{n}\left\| \frac{1}{\sqrt{s}} Rp'_i - \sqrt{s}q'_i \right\|^2 \tag{6.37}$$

the solution for *s* an be found to be

$$s = \sqrt{\frac{\sum_{i=1}^{n}\|p'_i\|^2}{\sum_{i=1}^{n}\|q'_i\|^2}} \tag{6.38}$$

The rotation matrix of the pose is independent of *s*, yet it reduces the overall least-squares objective function. After *R* and *s* are determined, *t* can be computed as:

$$t = s\overline{\mathbf{v}} - R\overline{p} \tag{6.39}$$

where $\overline{\mathbf{v}} = \frac{1}{n}\sum_{i=1}^{n}\hat{\mathbf{v}}$. Note that if the origin of the object space is placed at $\overline{p}$, i.e., $\overline{p} = 0$, then $s = t_z$.

6.2.5.2 Initial absolute orientation solution

With the **OI** algorithm, the initial rotation will be the same as those computed using the aforementioned weak-perspective algorithm, however, the translation is different in that it is computed using (6.19) as a result of optimizing (6.18). Let us rewrite (6.19) here

$$t(R) = \frac{1}{n}\left(I - \frac{1}{n}\sum_j \hat{V}_j\right)^{-1}\sum_j(\hat{V}_j - I)Rp_j \tag{6.40}$$

Comparing (6.39) and (6.40), we find that the former is approximated by the latter if the following conditions hold:

$$\left(I - \frac{1}{n}\sum_j \hat{V}_j\right)^{-1} \approx I \tag{6.41}$$

$$\frac{1}{n}\sum_j \hat{V}_j Rp_j \approx s\overline{\mathbf{v}} \tag{6.40}$$

The first condition states that the scene points are located close to the optical axis and the second condition states that the scene points are distributed like a plane parallel to the image plane. These two conditions closely resemble the conditions under which weak-perspective approximation is valid.

In summary, we have reformulated the pose estimation problem under the weak-perspective model as the problem of fitting the set of the reference points to a planar projection of the image points. Using the image points themselves as the hypothesized scene points in the initial absolute orientation iteration results in a pose solution better than the unmodified weak-perspective solution. This pose solution serves, therefore, as a good initial guess for the subsequent iterative refinement.

6.3 The LHM implementation

In the LHM implementation we use the same strategy illustrated in the end of the chapter 5. We have variable number of visible markers, that is only the positions of the detected markers are introduced in the internal function that provides the pose estimation. In the LHM implementation, we can choose to solve the pose estimation using the SVD method or the Quaternion method. Several simulation experiments showed that the SVD method is faster then Quaternion method. The algorithm is able to estimate the position with a minimum number of detected markers of 5. The internal iterations of the LHM algorithm are stopped in two cases, if the sum of squared object space error in the current iteration is less or equal to the parameter of the function ε or if the $(new\_err - old\_err)/old\_err$ is less or equal to a second parameter *Tolerance*. Where *new_err* and *old_err* are the sum of squared object space error in the current iteration and in the past iteration. In a nominal case the parameter ε and *Tolerance* are equal to $1*10^{-8}$ and $1*10^{-5}$ respectively. The tests on the pose estimation algorithm are provided in the next chapter.

7. PERFORMANCE OF GLSDC AND LHM

Pose estimation from a camera image is becoming a very used method to acquire relative position and orientation information. Any practical realization requires that the algorithm has certain characteristics (performance criteria). These performance criteria often depend upon the system that we have to control and its hardware. Performance criteria used in [5] are the time of simulation with a different number of point, robustness against noise in the image, quality of estimation depending on both the number of points, and effective field of view. In the Ansar and Daniilidis paper [5] the analysis is performed on static images. In this section we propose a different approach, that is we analyze the algorithms on the Autonomous Arial Refueling (AAR) simulation, this allows us to examine the algorithms in two cases: Simulated Vision (SV), where we have a fixed number of markers and input data are smooth, and Real Vision (RV) where we have a variable number of markers and the input data are characterized by extreme variability and sometimes they are not suitable for the pose estimation algorithm. We establish now the performance criteria to compare the algorithms:

1. Number of FLOPS.
2. Speed performance.
3. Estimated delay.
4. Differences between "true" values and estimated values and R.M.S. of the error.
5. Robustness on markers noisy position, labeling error and initial conditions.
6. Error propagation analysis.

The purpose of the following tests is to try to analyze the algorithms on many aspects that are code heaviness, delays, estimation consistency, robustness and error propagation.
The entire tests are performed with Matlab® 6.5 and Simulink® 5.0.

7.1 Number of FLOPS

The term FLOPS is a short for ***fl**oating-point **op**eration**s***, is a common benchmark measurement for rating the computational weight of one executed function. Floating point operation includes any operations that involve fractional numbers. Such operations, which take much longer to execute than integer operations, occur often in some applications. Most modern microprocessors include a floating-point unit (FPU), which is a specialized part of the microprocessor responsible for executing floating-point operations. The FLOPS measurement, therefore, measures the number of operation executed by the FPU, this is a good benchmark even if the computer could have some application resident into the memory that use the FPU in the meantime the tests are performed.

To extract the data for this benchmark we used Matlab® 5.3 because in the following Matlab versions the function *flops* is disabled. The data are summarized in table 7.1.

Simulated Vision		**Real Vision**	
GLSDC	LHM	GLSDC	LHM
19949385	12051938	14222319	21019524

Tab. 7.1: Number of FLOPS of the two algorithms in SV case and in RV case

Note that the GLSDC algorithm has a number of FLOPS bigger in the SV case than in the RV case, this happens because this algorithm estimates the position always in the same way with a fixed number of iterations (which in the nominal case is 4). In the SV case, the algorithm works always with the maximum number of markers, which means that the working matrix has the largest size, which in turn means that more flops are required to invert the matrix.

In the RV case instead, the size of the working matrix is limited by the number of seen markers, so the matrix size is smaller and it takes less flops to invert it.

The LHM algorithm process the data in a different way. In the SV case the incoming data are very smooth so the LHM algorithm is able to estimate the position using a relatively small number of steps, because the current rotation matrix R is very close to the current one.

In the RV case the data are noisy and irregular so the LHM has to use more steps to converge to a solution.

7.2 Speed performance

The speed performance is a simple test that gives an idea of the heaviness of the function. The test is obviously dependent from the utilized system resources and from the number of applications resident in memory, but if the measures are performed approximately in the same time, and with the same computer, the end result can be compared. If we join the speed performance data with FLOPS data we can have a better understanding of the computational heaviness of the functions. For this test we used a Pentium 4, 2.53 GHz laptop with 448 Mbytes of RAM. We measured the speed performance with the *profiler* tool of Matlab®. This tool gives the running time in seconds for each called function and sub-function. The test consists in the execution of the 2 algorithms in the SV case and in RV case. The simulation lasted 40 seconds, a sampling time of 0.1 sec was used by the MV algorithms. At the end of the test the estimation algorithm is called 4010 times, in this way we have a good estimation of the execution mean time for each function.

The profiler tool provides a result called *Time/call*, this measures the mean time of execution for the function.

Simulated Vision		**Real Vision**	
GLSDC (sec)	LHM (sec)	GLSDC (sec)	LHM (sec)
0.0053910447761	0.0158049751244	0.0053000000000	0.02006865671642

Tab. 7.2: Mean time of execution for GLSDC and LHM algorithms in SV and RV case

Let us analyze the data in Tab. 7.2. In the SV case although the GLSDC has a grater number of FLOPS, it has a lesser mean time of execution, and if we calculate the ratio from the two data we obtain 2.931. This means that the LHM function can use up to triple the GLSLD time. We can see that the GLSDC algorithm uses around the same time in SV case and RV case, unexpectedly from the FLOPS results, while the LHM algorithm uses more time in the RV case as it was to be expected, we can see that the ratio in RV case grows up to 3.786. Moreover, we can see that the differences of mean time of execution is not very relevant, in the LHM algorithm, between SV case and RV case as it occur in the FLOPS case instead.

7.3 Estimated delay

The estimation of the delay is a test in the SV case that consists searching the time delay that minimizes the error between the "real" value (value obtained by the sensor without GPS noise) and the estimated value of the vision algorithm. This prove is obviously connected to the precision of camera parameter, and in particular to both the *field of view* (*fov*) parameter and the

GPS system, because the SV case gives one estimation of the marker position based on the measure of GPS system (see chapter 4 and 3). The estimated delay is not therefore a parameter tightly correlated to the algorithm but if we compare the two obtained values, and the values are obtained in the same condition, we can decide which algorithm provides first the measure. For the search of the estimated delay we can make a simple function on Matlab® that translates in time the distance found in SV case (*x, y, z*) and calculates the instant time when we have a minimum error between this value and the estimated value. The function uses the Frobenius norm applied to the error matrix *e,* which is difference between value in SV case and "real" value, for calculate the root mean square (*rms*):

$$rms = \frac{\|e\|_F}{\sqrt{n}} = \frac{\sqrt{\sum diag(e^T e)}}{\sqrt{n}} \tag{7.1}$$

we search the value of the time t than minimizes the *rms* function delaying the distance in SV case. We found that the estimated delay for the GLSDC algorithm is *t* = 0.3 sec, while for the LHM algorithm is *t* = 0.25 sec. An example is reported in fig. 7.1 where we can see that the GLSDC line is a better estimation of the red line rather than the blue line.

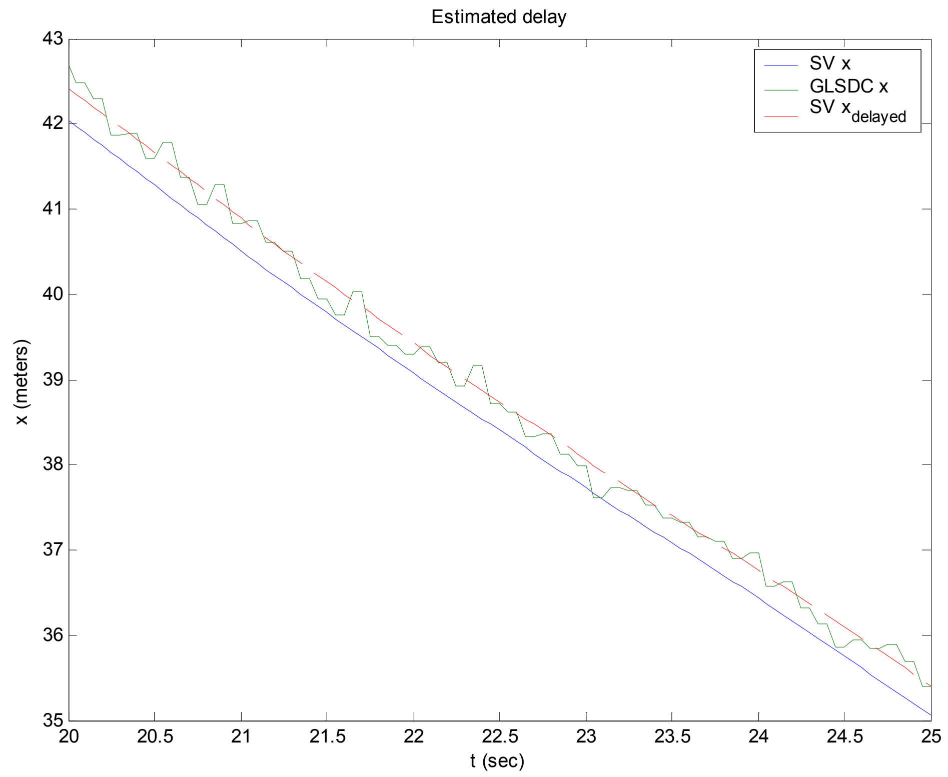


Fig. 7.1: Estimated delay for GLSDC (t = 0.3 sec)

7.4 Differences between true values and estimated values

Before comparing estimated values with true ones, we have to clarify that "true values" are defined as the distance and orientation of the tanker in camera frame, calculated using the readings from the linear and angular position (simulated) sensors. In practice a white noise having power spectral density of p=10^{-9} and covariance of c=$2*10^{-8}$ is added to the Euler angles (roll, pitch, yaw) sensors to simulate sensor noise. These signals are used to calculate the homogeneous transformation matrix ${}^{C}T_{T}$ with the relative distance *(x, y, z)* between camera and tanker. We examine the differences in SV case and RV case keeping in mind that the true values are different for LHM and GLSDC algorithm because they are dependent by the trajectory that the UAV cross that is in relationship with the estimated distance. We can see in fig. 7.2, 7.3, 7.4, 7.5 the difference between GLSDC and true values in SV case.

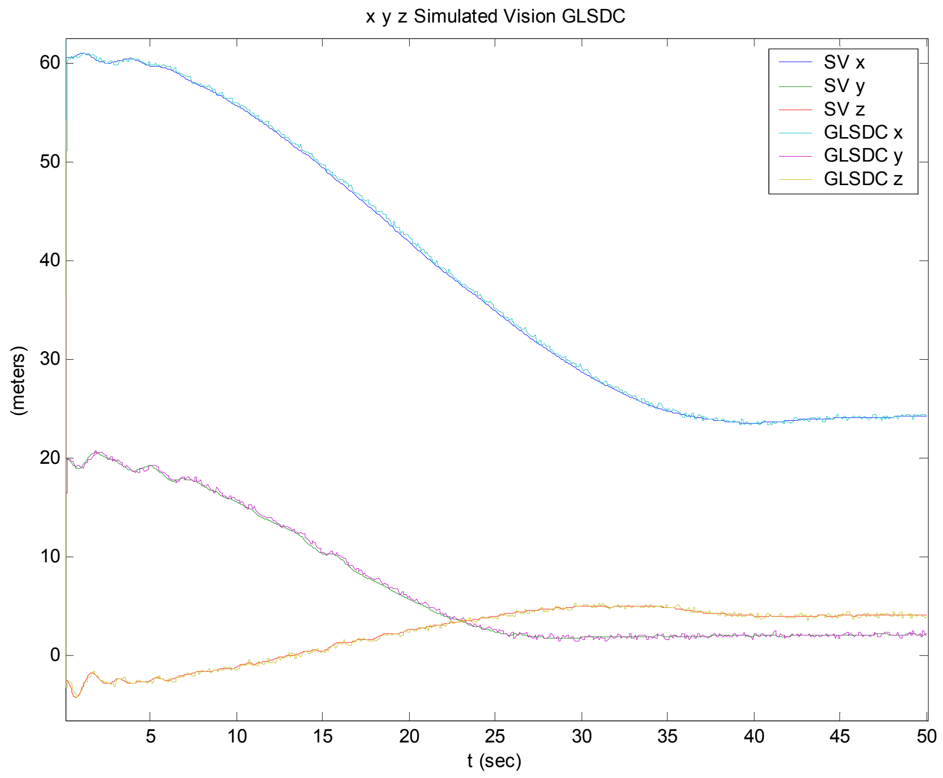


Fig. 7.2: differences between real x y z and GLSDC x y z in SV case

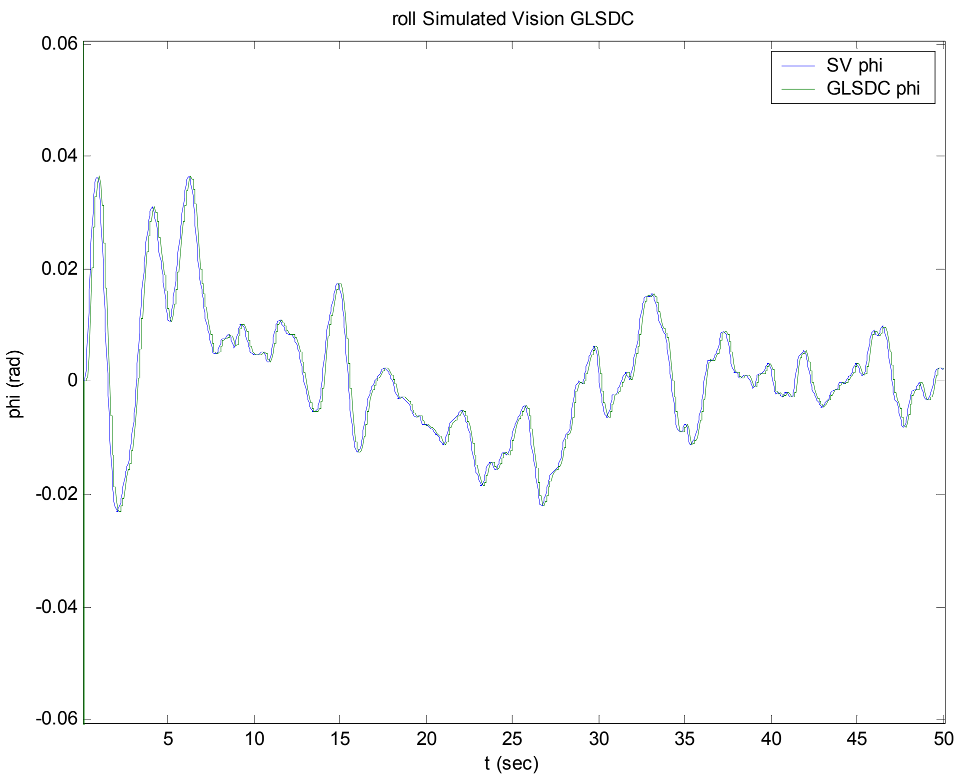


Fig. 7.3: difference between real roll and GLSDC roll in SV case

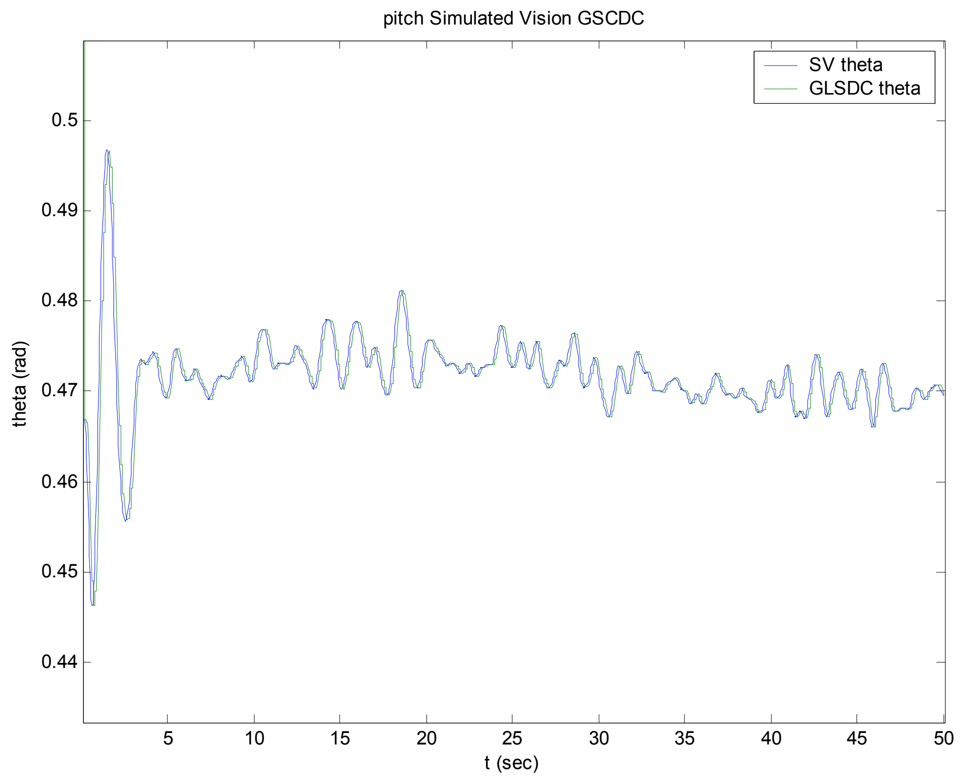


Fig. 7.4: difference between real pitch and GLSDC pitch in SV case

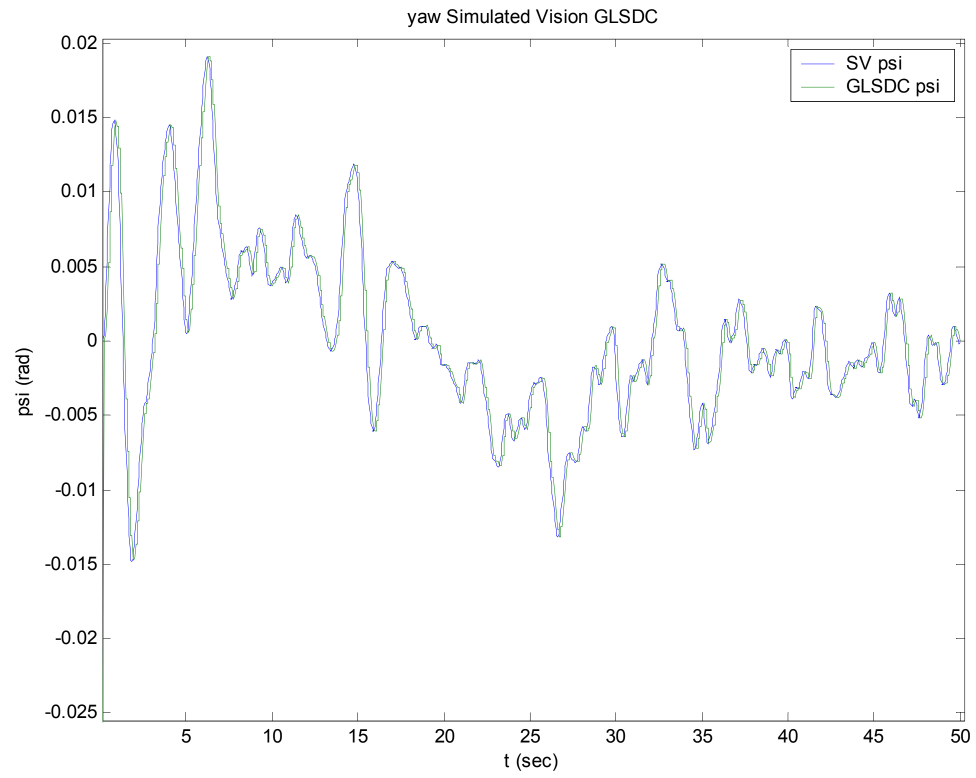


Fig. 7.5: difference between real yaw and GLSDC yaw in SV case

In fig. 7.6, 7.7, 7.8, 7.9 we represent the same data for LHM algorithm

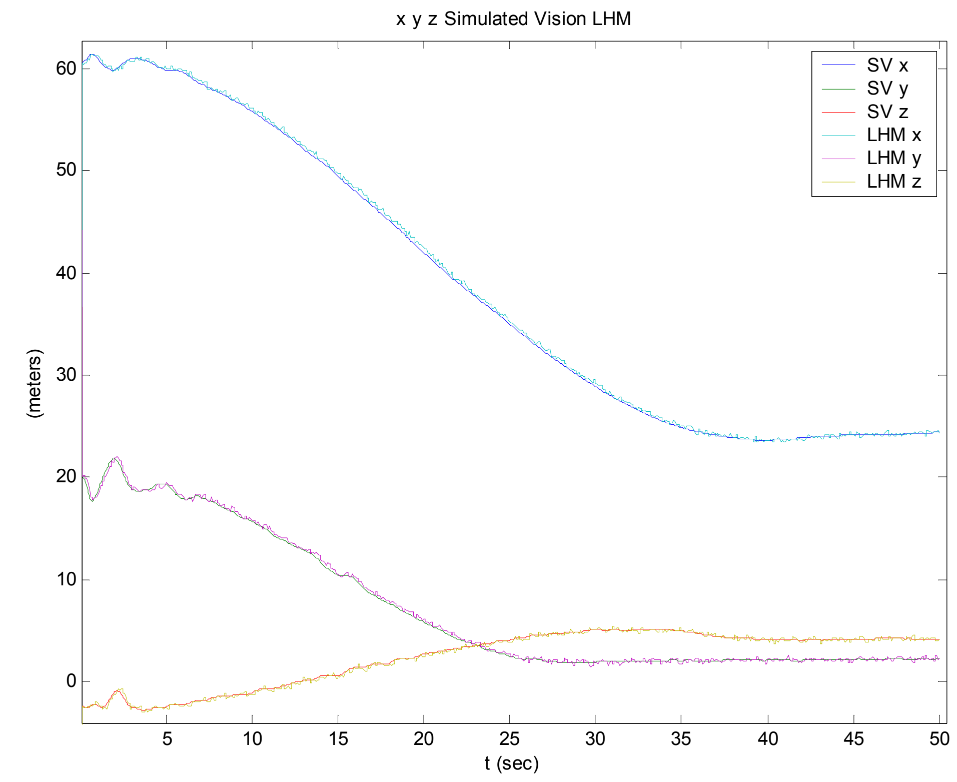


Fig. 7.6: differences between real x y z and LHM x y z in SV case

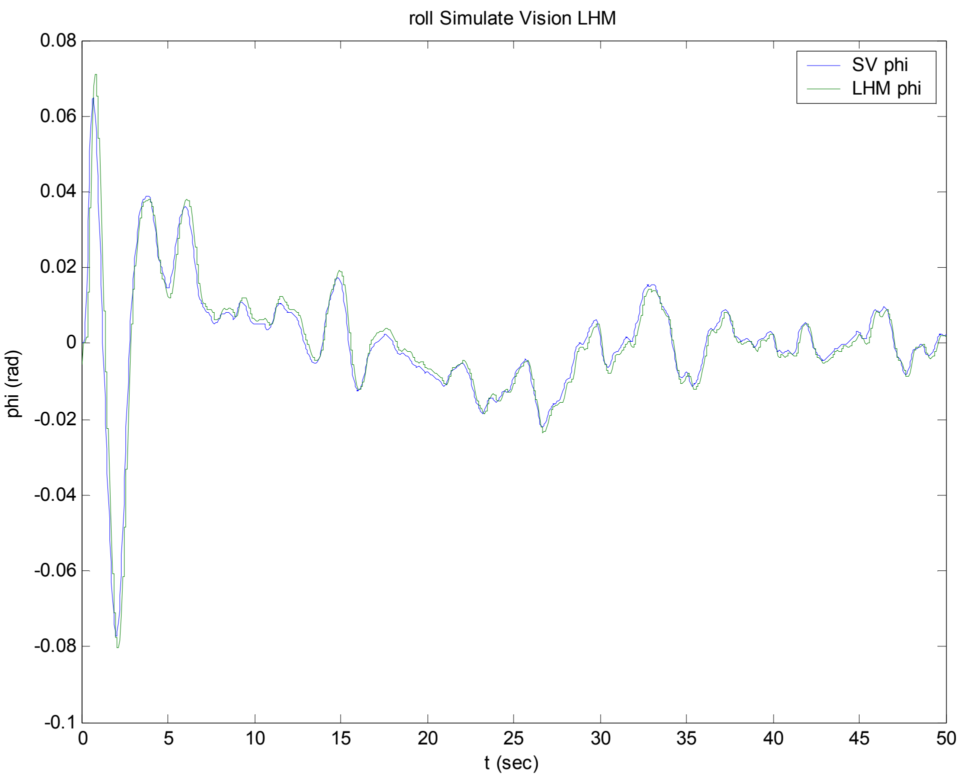


Fig. 7.7: difference between real roll and LHM roll in SV case

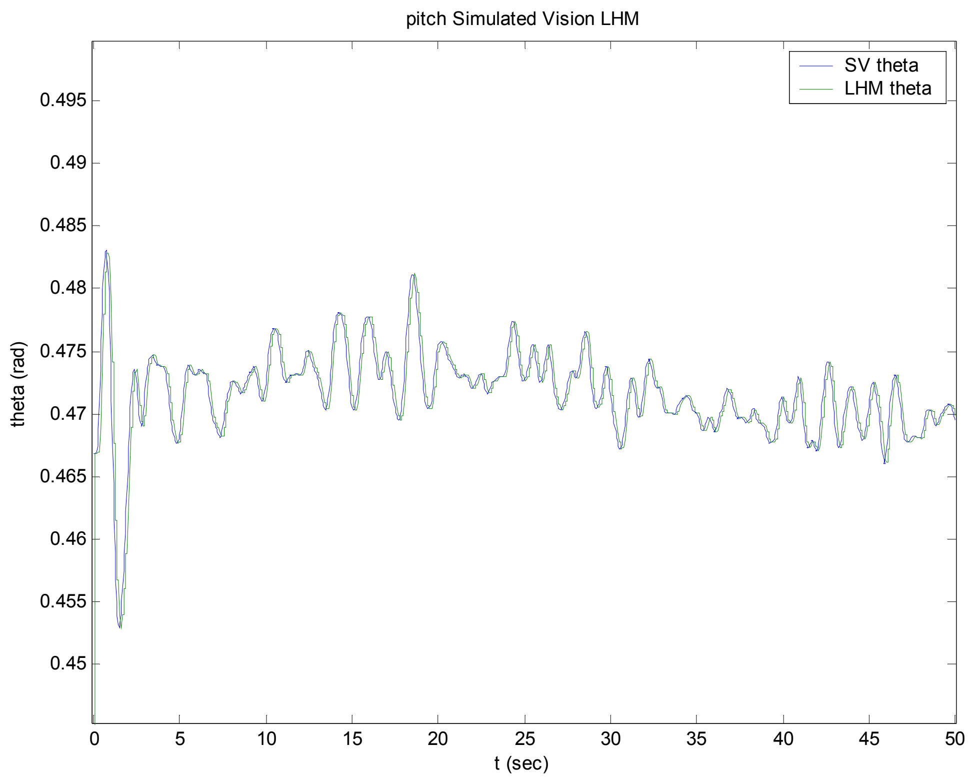


Fig. 7.8: difference between real pitch and LHM pitch in SV case

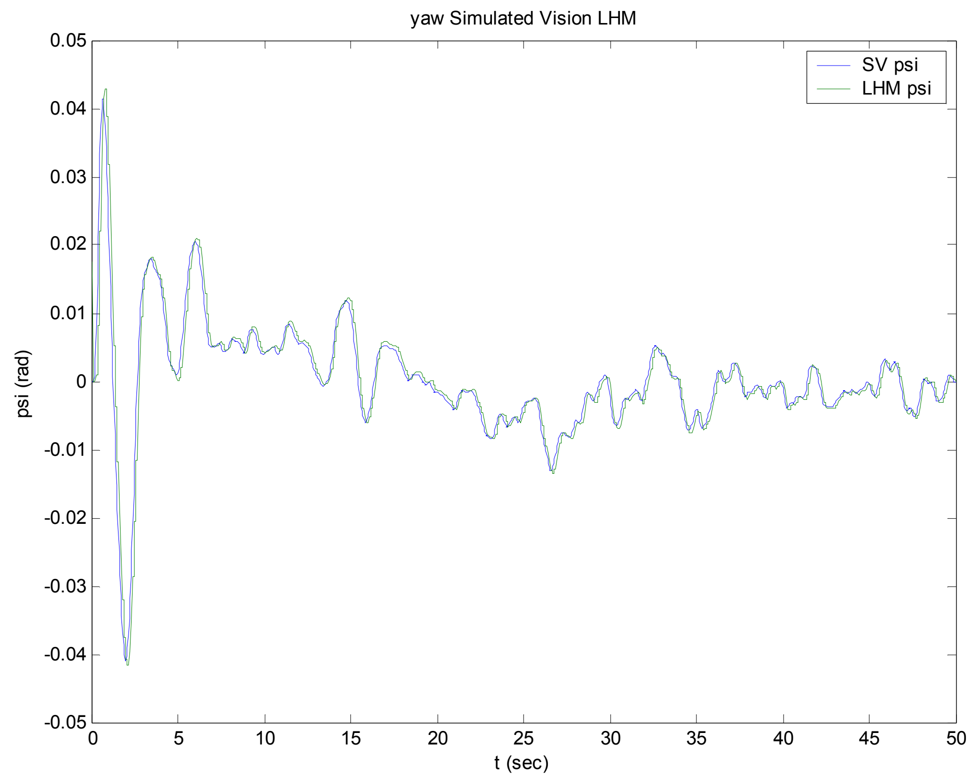


Fig. 7.9: difference between real yaw and LHM yaw in SV case

It is apparent form the above graphs that the difference is negligible. Therefore, other parameters of comparison are needed in order to chose between the two algorithms. Using the *rms* (7.1) where *e* is now a vector and the Frobenius norm is therefore equivalent to the norm 2, we obtain the following data:

	X	**Y**	**Z**	**Roll**	**Pitch**	**Yaw**
GLSDC	1.7042	0.9291	3.0024	0.0500	0.0127	0.0438
LHM	1.2936	1.2955	0.8005	0.0039	0.0214	0.0025

Tab. 7.3: rms values of the error for GLSDC and LHM algorithm in SV case

Analyzing the data in Tab. 7.3 we can see that the LHM algorithm has better estimation for the variable *(x, z, Roll, Yaw),* the larger difference is in the variable *z* which is one of the most important variables. In fact, within the AAR problem, Euler angles are provided by the gyro, therefore the quality of the Euler angles provided by the Pose Estimation algorithm is not of primary importance. Conversely, the accuracy of the distance information is of primary concern.

We proceed now with the comparison of the graphs in RV case, this is very important to understand the characteristics of the data that an estimation algorithm provides, which is in turn essential for the design of a controller based on machine vision.

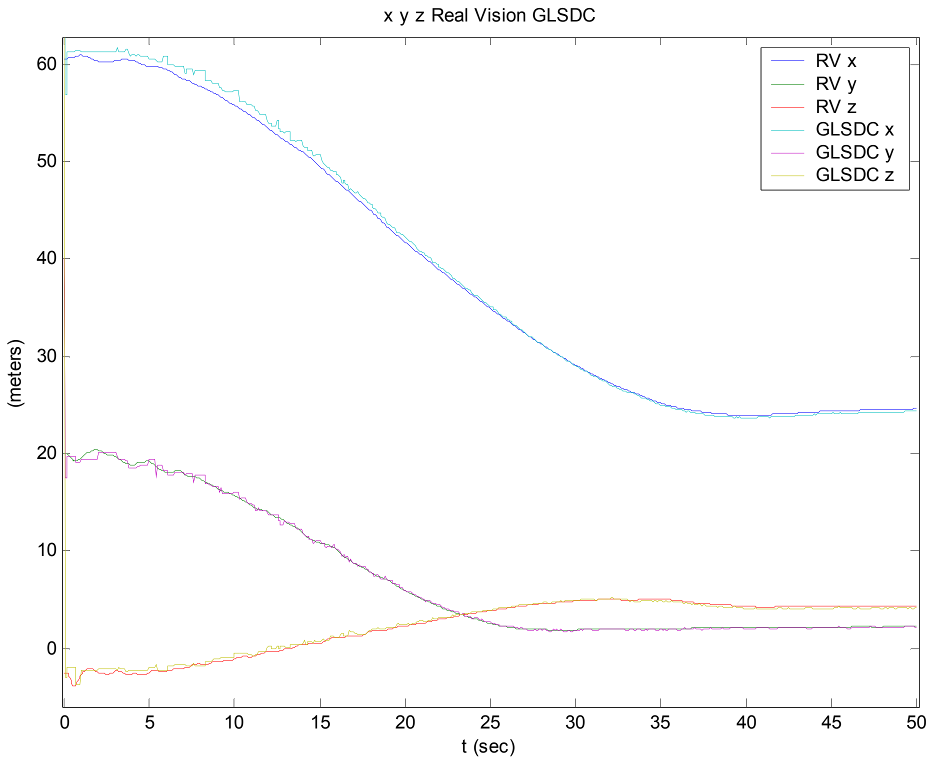


Fig. 7.10: difference between real x y z and GLSDC x y z in RV case

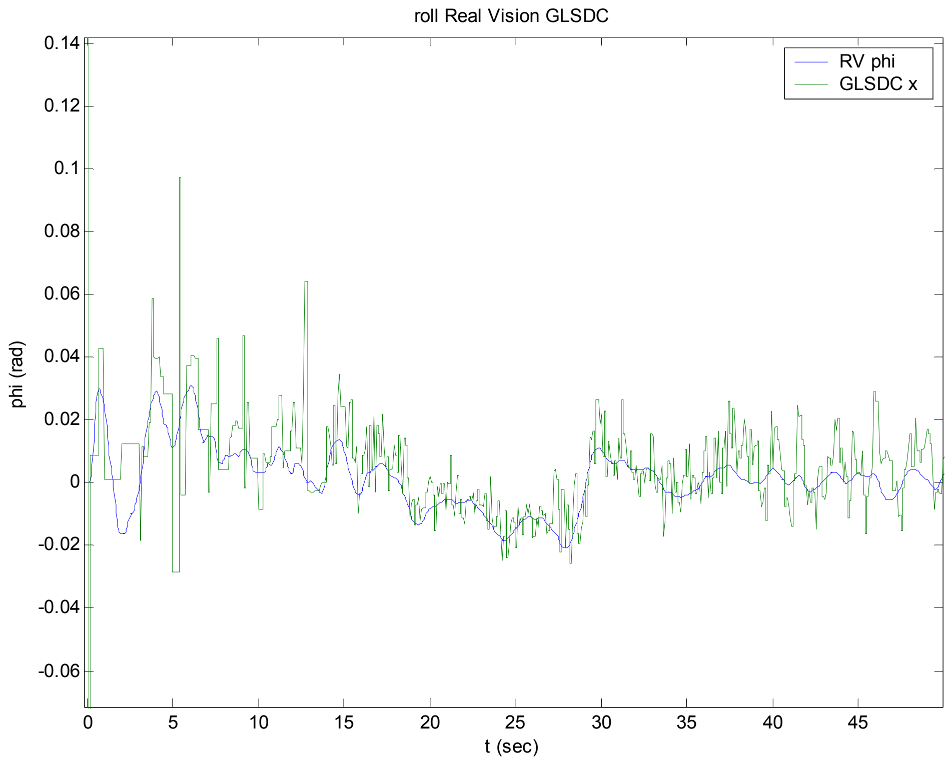


Fig. 7.11: difference between real roll and GLSDC roll in RV case

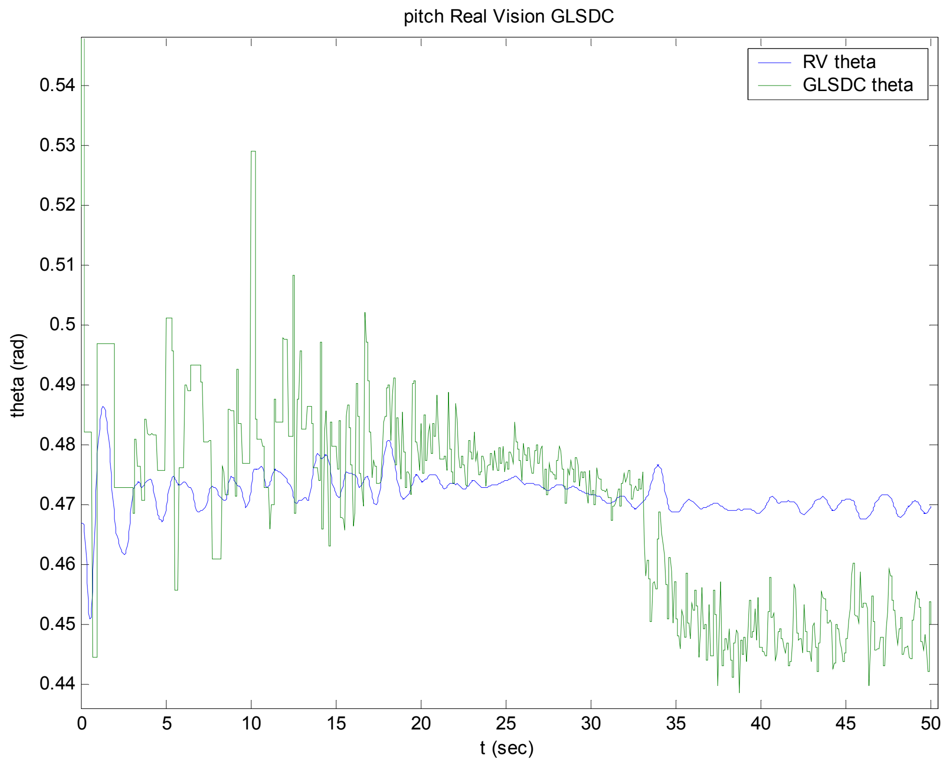


Fig. 7.12: difference between real pitch and GLSDC pitch in RV case

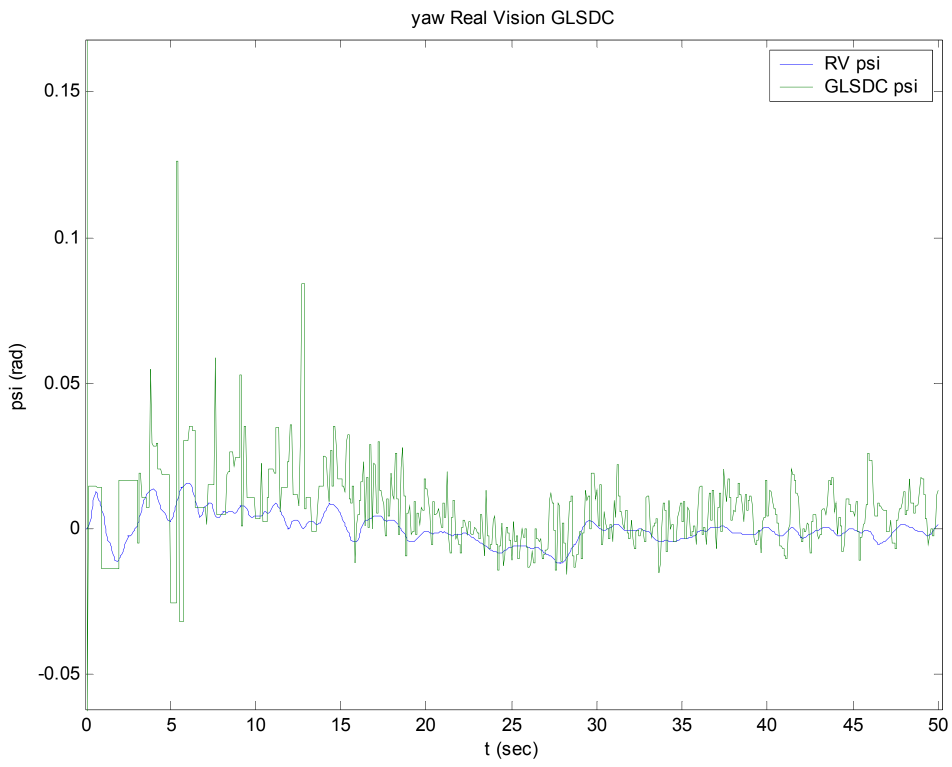


Fig. 7.13: difference between real yaw and GLSDC yaw in RV case

We can see from fig. 7.10 – 7.13 that the results that GLSDC algorithm provides in the RV case are very different from the results obtained in the SV case. Since these results are dependent of the number of seen markers, in fig 7.14 – 7.17, we present the data for the LHM algorithm only in the RV case.

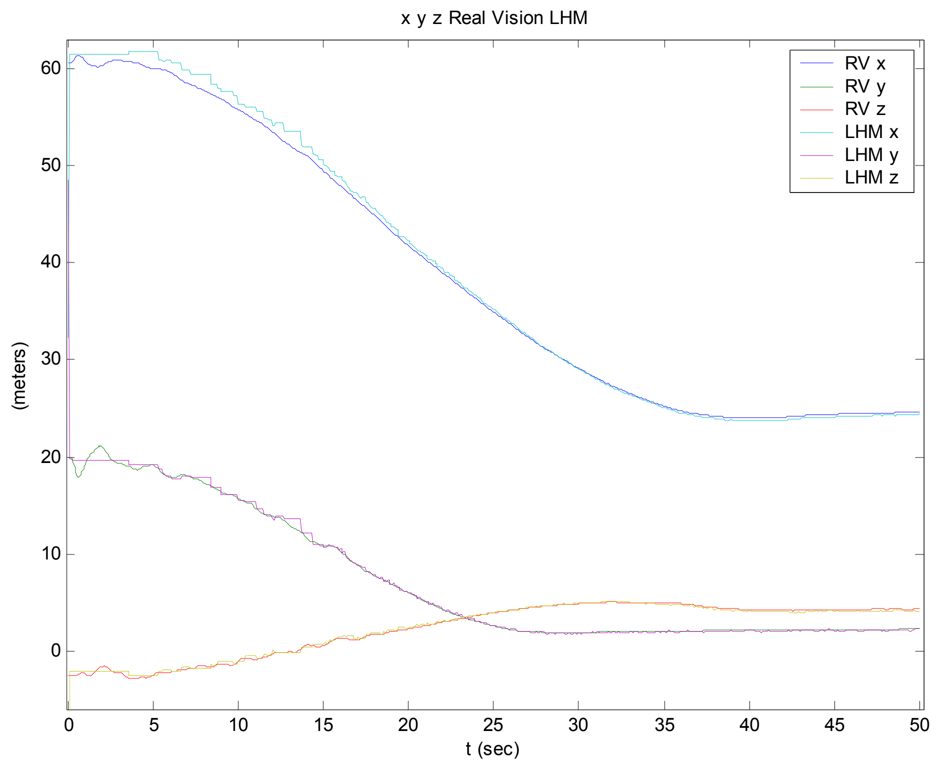


Fig. 7.14: difference between real x y z and LHM x y z in RV case

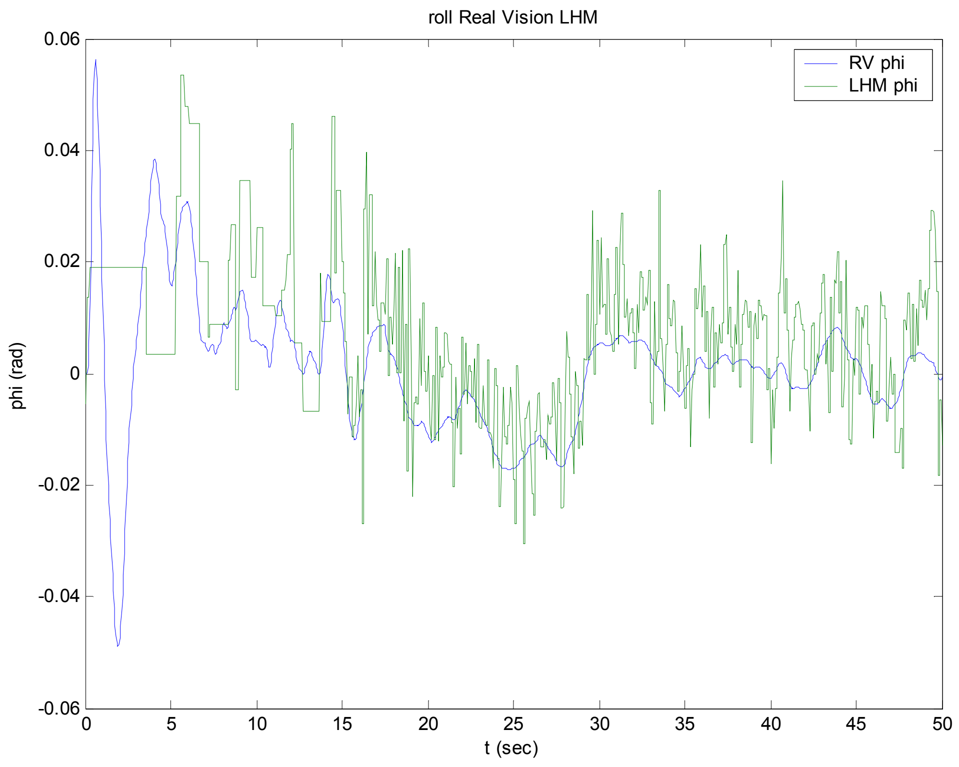


Fig. 7.15: difference between real roll and LHM roll in RV case

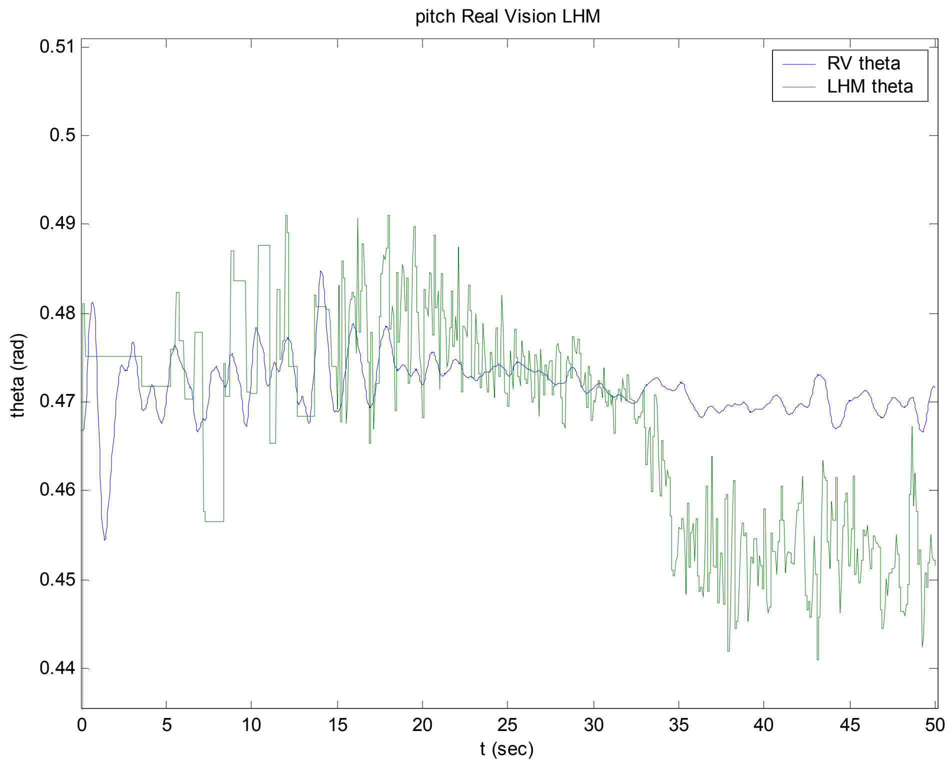


Fig. 7.16: difference between real pitch and LHM pitch in RV case

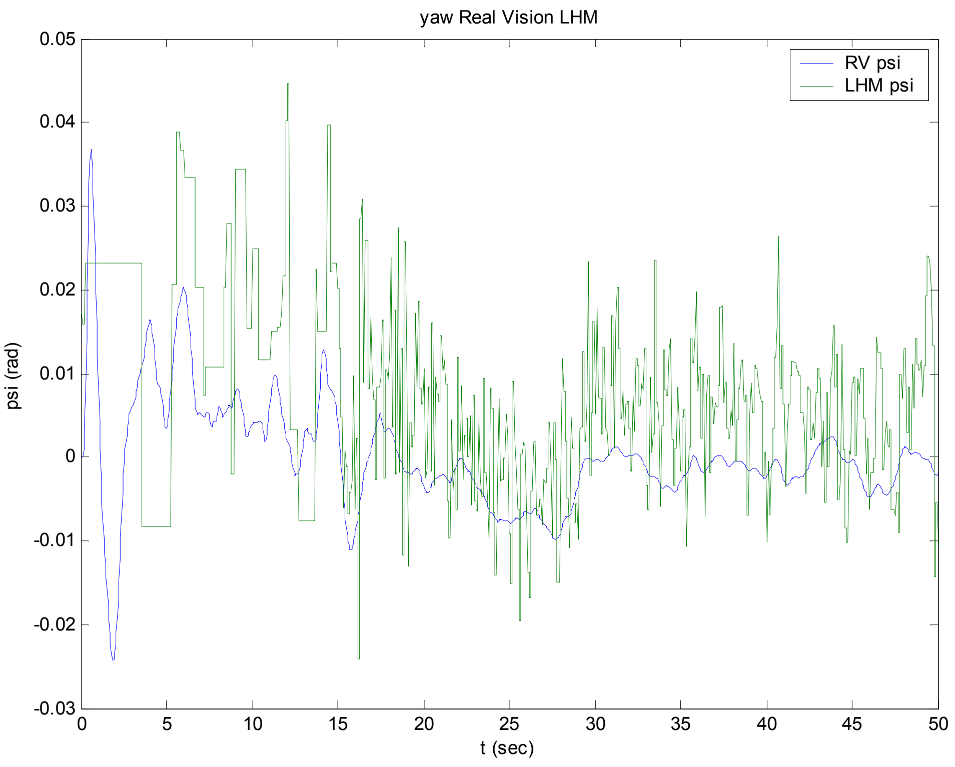


Fig. 7.17: difference between real yaw and LHM yaw in RV case

Looking the Fig. 7.11 – 7.13 and Fig. 7.15 – 7.17 we see that the Euler angles are very noisy and these are not fit to be insert in a loop system without an apposite filter.

Now we can analyze the data in RV case using the *rms* (7.1) analysis as we have done in the SV case, and we obtain:

	X	Y	Z	Roll	Pitch	Yaw
GLSDC	1.7395	0.9225	3.0089	0.0488	0.0193	0.0444
LHM	1.4474	1.3117	0.8040	0.0156	0.0242	0.0137

Tab. 7.4: rms values of the error for GLSDC and LHM algorithm in RV case

Analyzing the obtained result for the RV case we can make the same considerations done for Tab. 7.4 but now we observe that the LHM needs a minimum of 5 markers for have a good estimation and the GLSDC need 4 markers and moreover the GLSDC have initial data strongly dependent from the initial condition, we decide, therefore, to start the comparison when all algorithm work in optimal situation and we extract from the data in RV case one table with the start data for t_1 = 15 sec and end data t_2 = 50 sec, we obtain:

	X	Y	Z	Roll	Pitch	Yaw
GLSDC	0.3254	0.0962	0.1679	0.0090	0.0150	0.0098
LHM	0.3364	0.0963	0.1367	0.0109	0.0125	0.0104

Tab. 7.5: rms values of the error for GLSDC and LHM algorithm in RV case between t_1=15 sec and t_2=50 sec

In Tab.7.5 we can see that the values are very similar and we can tell that the GLSDC and LHM are equivalent when enter in a good condition for work. These conditions, as described next, are different, so we have to apprise which one is better for the used simulation.

7.5 Robustness

In this section we analyze the robustness of the algorithm, for this we have to establish the method of comparison, we test the GLSDC and LHM algorithm in four cases:

- Noise addiction in the markers position with correct labeling.
- Error in the labeling.
- A real case: noise addiction in the markers position with uncertain labeling.
- Error in initial condition.

These tests try to show the behavior of the algorithm in different cases and therefore a design for one control more robust.

7.5.1 Noise addition in the markers position with correct labeling

This test consist in the addition of the noise at the position of the markers in the RV case after that the labeling function has been performed, one explanation is given in Fig. 7.18. The position is consequently the input of the estimation function, in this way we are sure that the algorithm performs the estimation on correct labeled data with noise, this is evidently different from a real case, since in reality noisy data are processed by both labeling function and pose estimation algorithm.

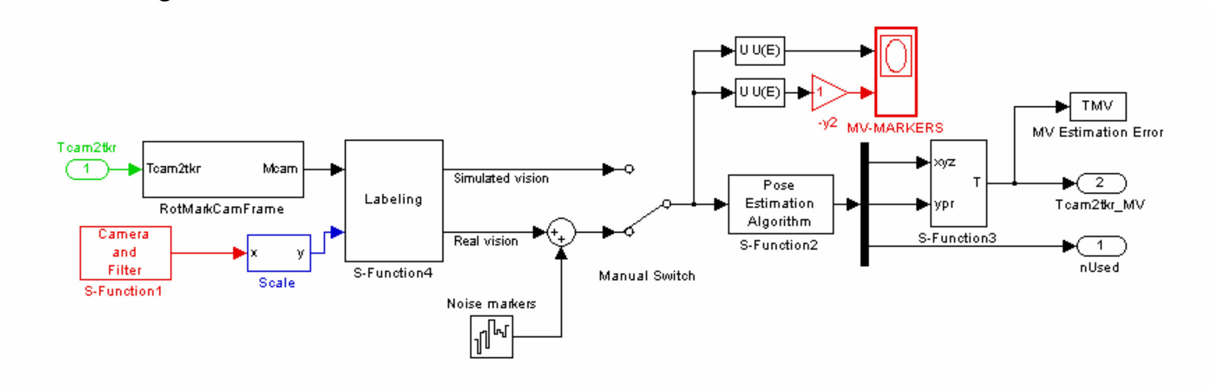


Fig. 7.18: noise adding after the labeling function

The test consist in the execution of the simulation with different noise power, in order to see clearly the behavior of the algorithms, the Machine Vision (MV) results do not enter in the control loop because the MV data are too noisy and can produce chattering effects in the control which can in turn bring instability. A real white noise has a correlation time of 0, a flat power spectral density (PSD) and a covariance of infinity, which are properties that cannot be achieved in reality but can only be approximated. Therefore, the used noise is a band limited white noise with correlation time t_c=0.05 with height of PSD equal to the parameter Noise power. The covariance is the Noise power divided by t_c. We have tested the algorithm with different Noise powers, starting form 0 to $5*10^{-9}$ with an interval of $1*10^{-9}$ and variable number of markers, as it happens in reality, the number of seen markers are visible in Fig 7.19 - 7.20 where we represent the number of seen markers for the all simulations.

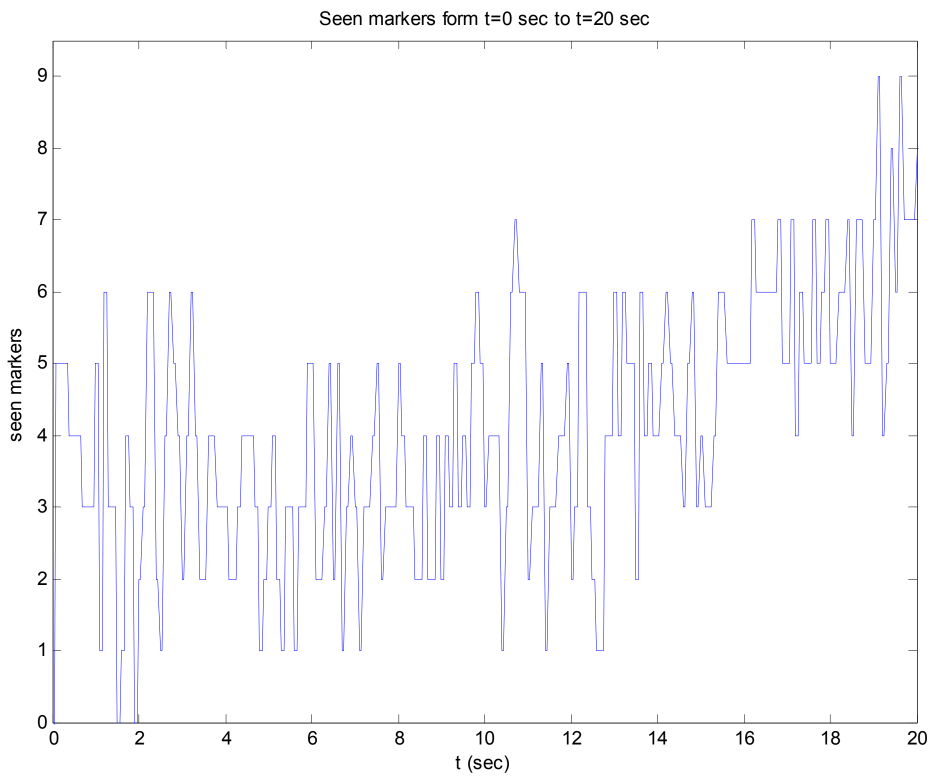


Fig. 7.19: seen markers from t=0 to t=20 sec

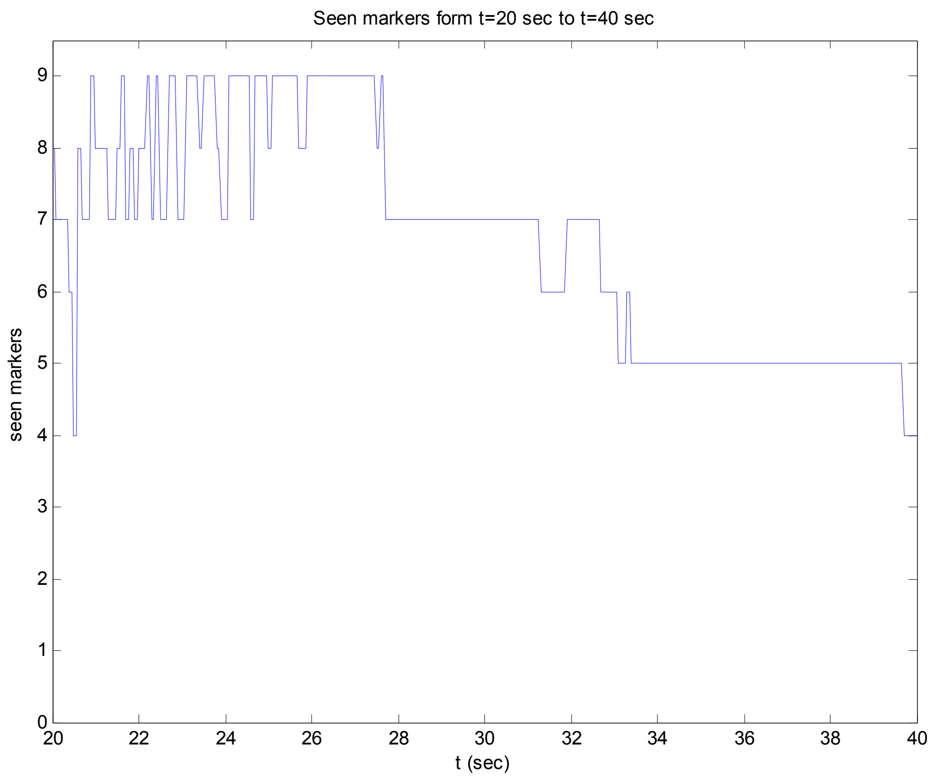


Fig. 7.20: seen markers from t=20 to t=40 sec

Whenever the algorithms work, they tend to produce similar results, however, it is interesting to observe the conditions in which they do not work and why this happens. We can see in Fig. 7.21 and 7.22 that for a noise power until $4*10^{-9}$ both algorithms do not have great problems, we can see that the estimation has an additional noise, (which is related to the noise power in input, this relationship will be analyzed later). For a noise power of $4*10^{-9}$ the LHM begins to show some problem as we can see in Fig. 7.23, that is, with low number of seen markers, the LHM algorithm can provide an incorrect estimation. In this case the estimation is provided with 5 markers, this problem shows up also for a noise power of $5*10^{-9}$ where in fact is more common.

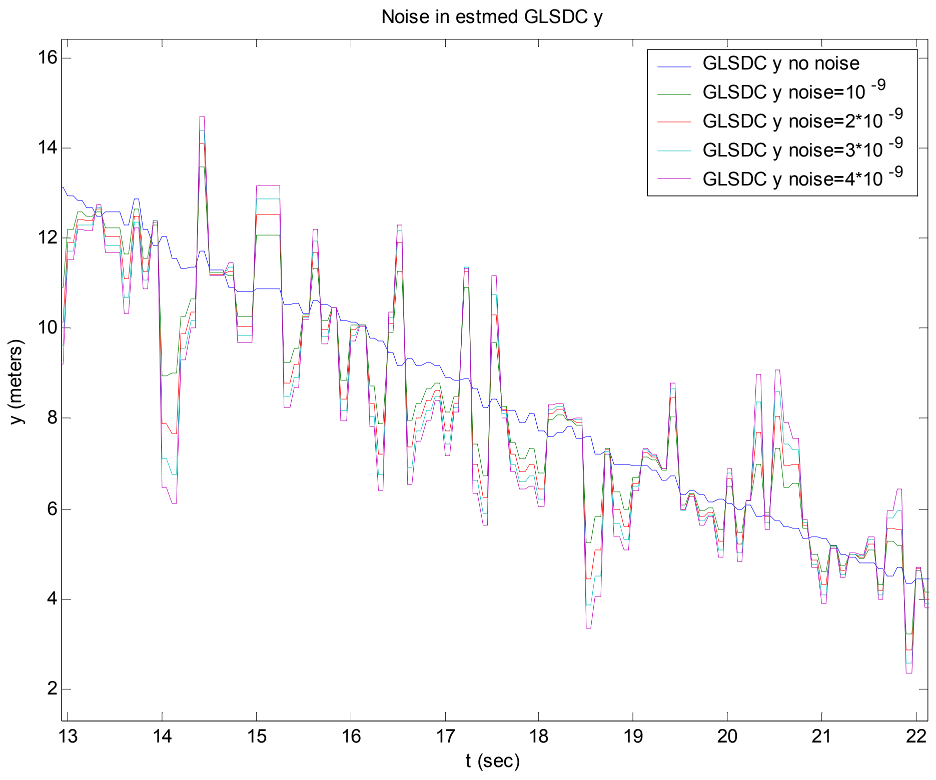


Fig. 7.21: effect of the noise in the GLSDC algorithm

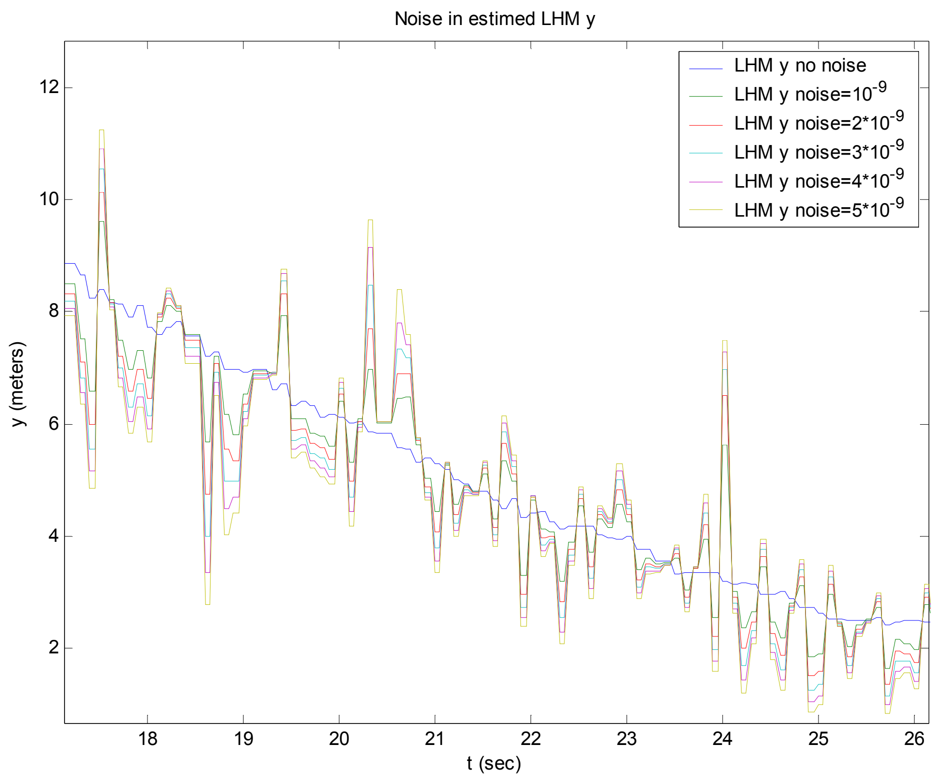


Fig. 7.22: effect of the noise in the LHM algorithm

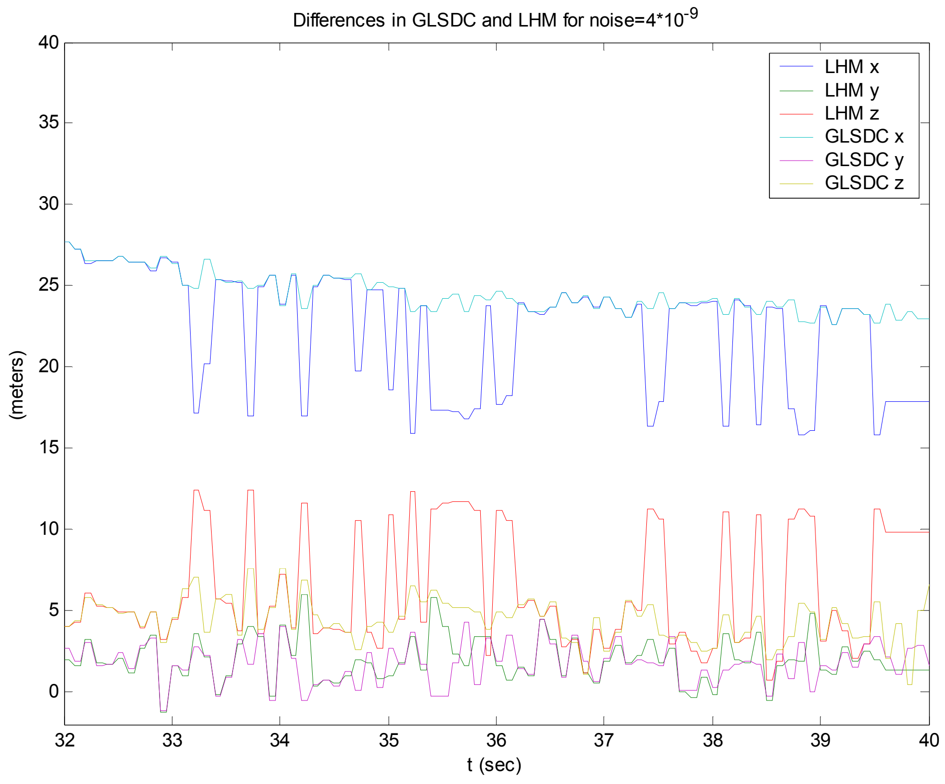


*Fig. 7.23: differences between GLSDC and LHM algorithm for a noise power =4*10^{-9}*

We have seen what happens in the LHM algorithm whenever the noise power is increased to $5*10^{-9}$ and beyond, that is sometimes when the number of seen markers decreases the LHM can provide an incorrect estimation. However, a correct estimation can be promptly recovered whenever the number of seen markers increases or the noise power decreases. It is important to notice that the above behavior is not shown by the GLSDC algorithm, indeed when the GLSDC algorithm stop providing a good estimation it often becomes unstable and there is no way of bringing it back to state that yields a good estimation. This fact is presented in Fig 7.24 and 7.25, where the behavior of the algorithms with a noise power at $5*10^{-9}$ is shown. We can see that the GLSDC algorithm starts with a good estimation, then at t=11.7 sec the algorithm restarts the estimation process since for a moment the number of seen markers goes below 4, at this point the estimated measures diverge and although the number of seen markers goes soon to 9 the algorithm does not come back to a state presenting a reasonable estimation. In Fig 7.25 we can see that the biggest problems for LHM algorithm happen for 27.7< t <30.7 sec and t >33 sec. In fact, analyzing this case, we can see that for t = 27.7 we have a number of seen markers that decreases from 9 to 7 and until t = 31.2 the numbers of markers is always 7. This decrease, together with a peak of noise, causes an incorrect estimation. Similarly, for t=33 sec we have a decrease of markers from 6 to 5 (that is the minimum number of markers for LHM algorithm) and this creates a problem for the estimation when the noise is too high.

A test was performed by fixing the numbers of markers at 9, the result is that the two algorithms evolve in a similar way and none of the two diverges or gives relevant errors, so it can be deduced that the better way to get a good robustness is to have a suitable number of markers for all the simulation time.

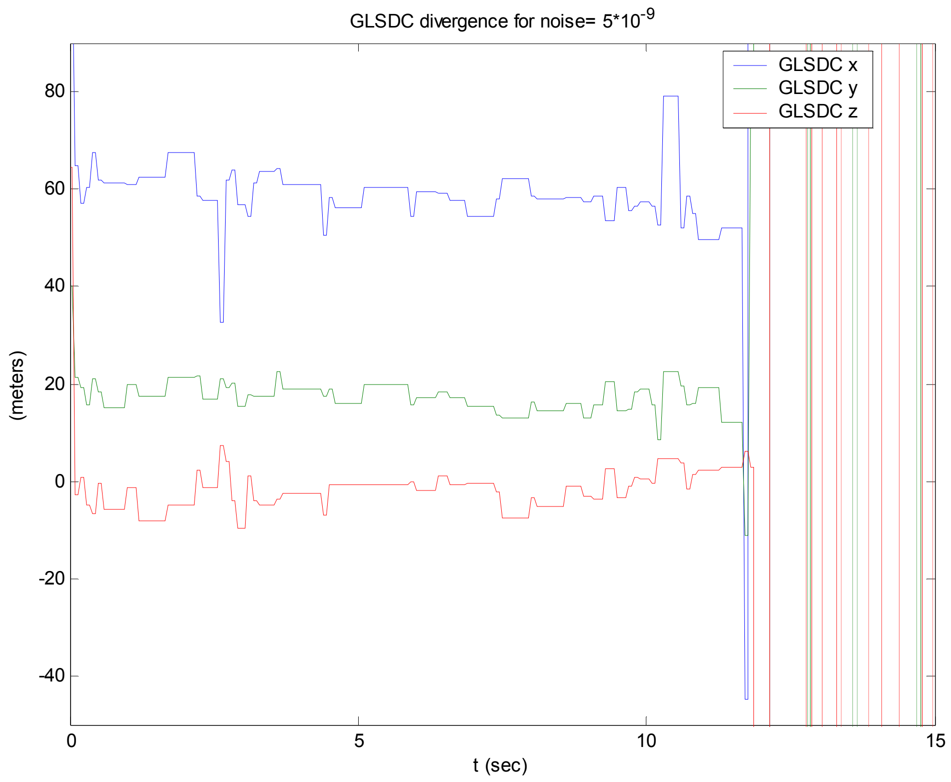


Fig. 7.24: GLSDC behavior with noise power $5*10^{-9}$, divergence for t=11.7 sec

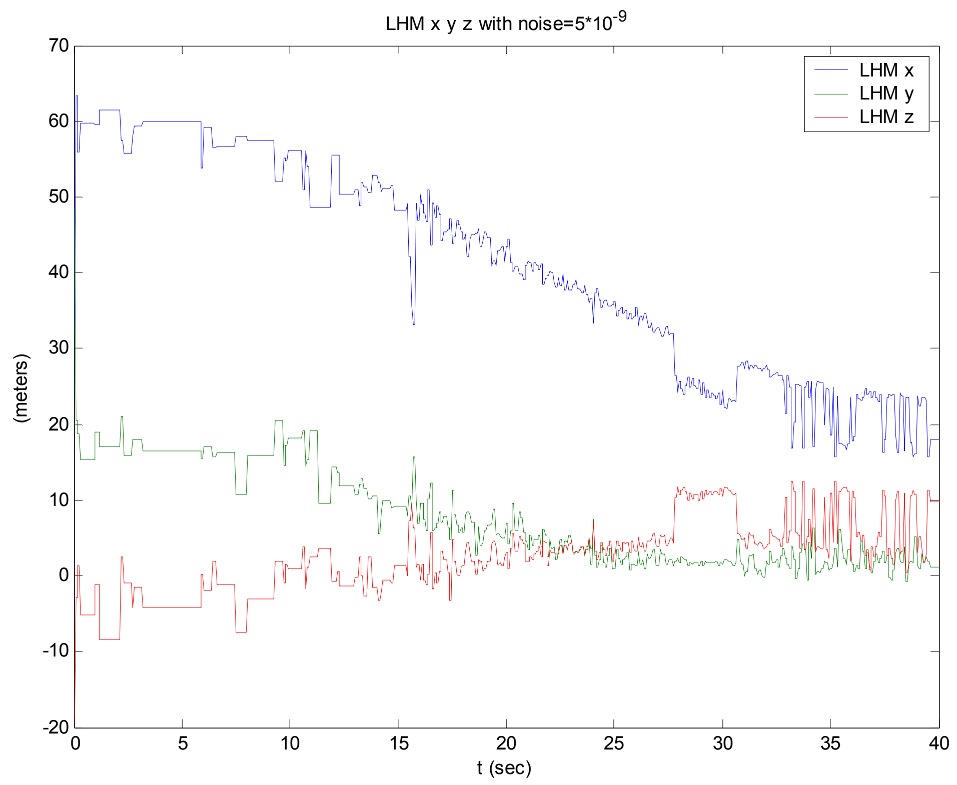


*Fig. 7.25: LHM behavior with noise power 5*10^{-9}*

7.5.2 Robustness to labeling errors

The test consists on the inversion of two markers in RV case when the MV does not enter in the control loop. This allows analyzing the behavior of the algorithms without holding in consideration the possible problems that would derive form using the MV estimation in the control loop. In the test we invert the found positions for the markers 1 and 2 for two execution of the pose estimation algorithm, specifically for t = 20 sec and t = 20.1 sec. During these time instants we see the markers $\{1, 2, 3, 5, 6, 7, 8, 9\}$ for the first call and $\{1, 2, 3, 4, 6, 7, 8\}$ in the second call of the estimation algorithms. In Fig. 7.26 we can see the markers positions (with relative identification number) on the tanker, in Fig. 7.27 and 7.28 we show the response of the two algorithms to the labeling error. The GLSDC algorithm suffers the error more than LHM, indeed the estimated x lowers form 42.8 to 12.6 against the 33.5 of the LHM, and all errors in the estimated variables are smaller for the LHM. Even more important is the fact that the GLSDC algorithm needs a considerable resumption time, in fact we can see that the measures return consistent only at time t=20.6 against the t=20.3 of the LHM, in other words, the first algorithm needs a settling time of 0.3 when we have a labeling error.

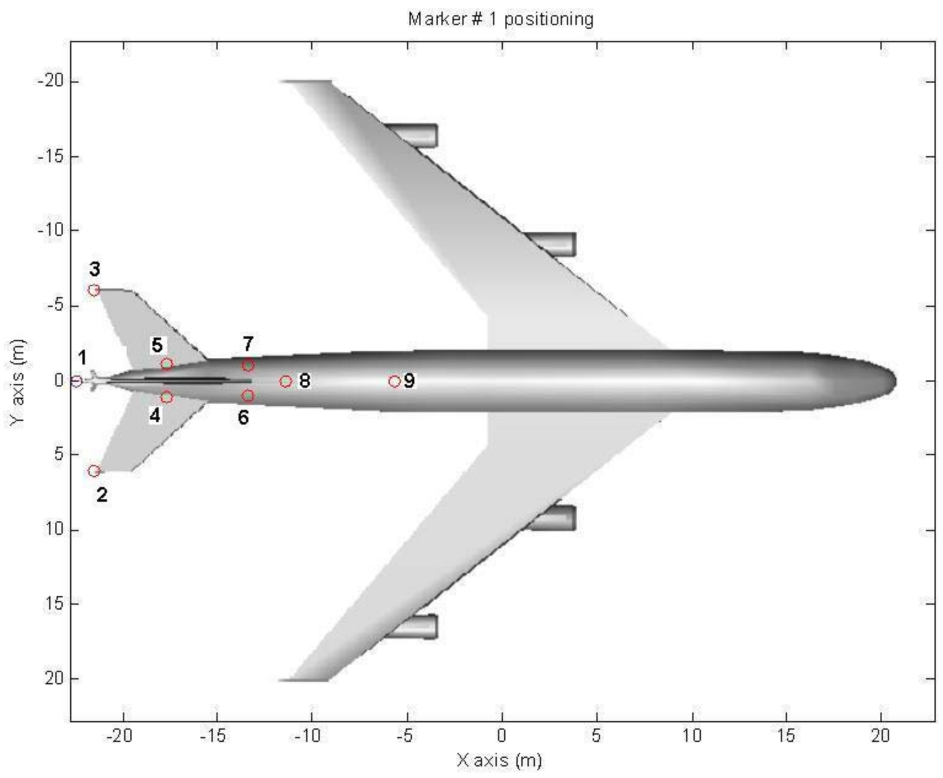


Fig. 7.26: Numbers and positions of markers on the tanker

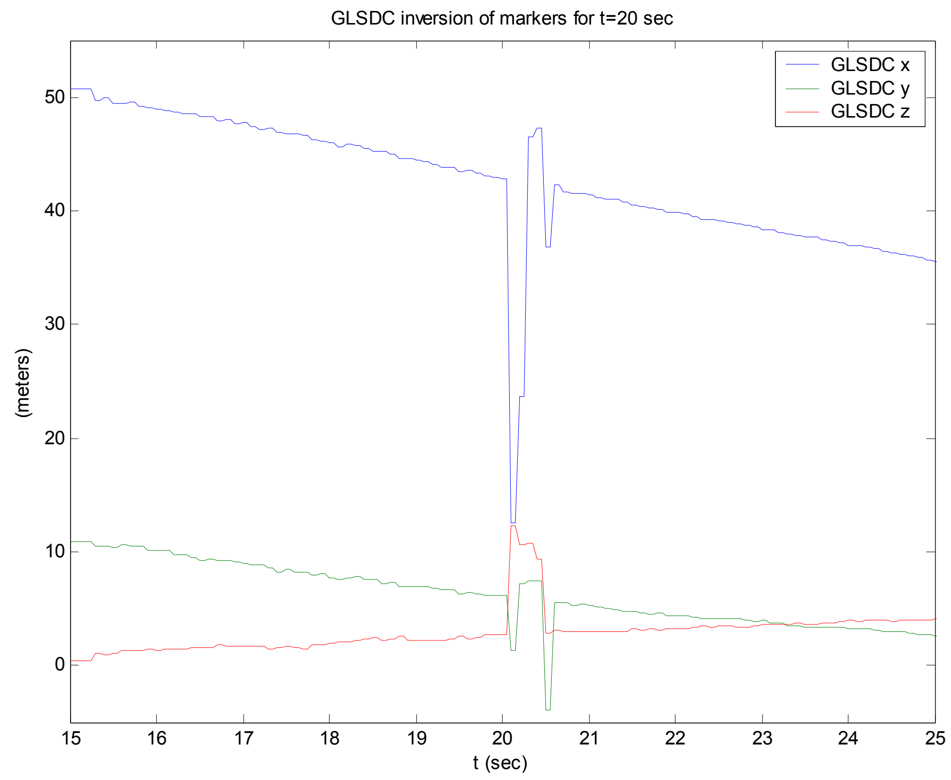


Fig. 7.27: labeling error response with GLSDC algorithm

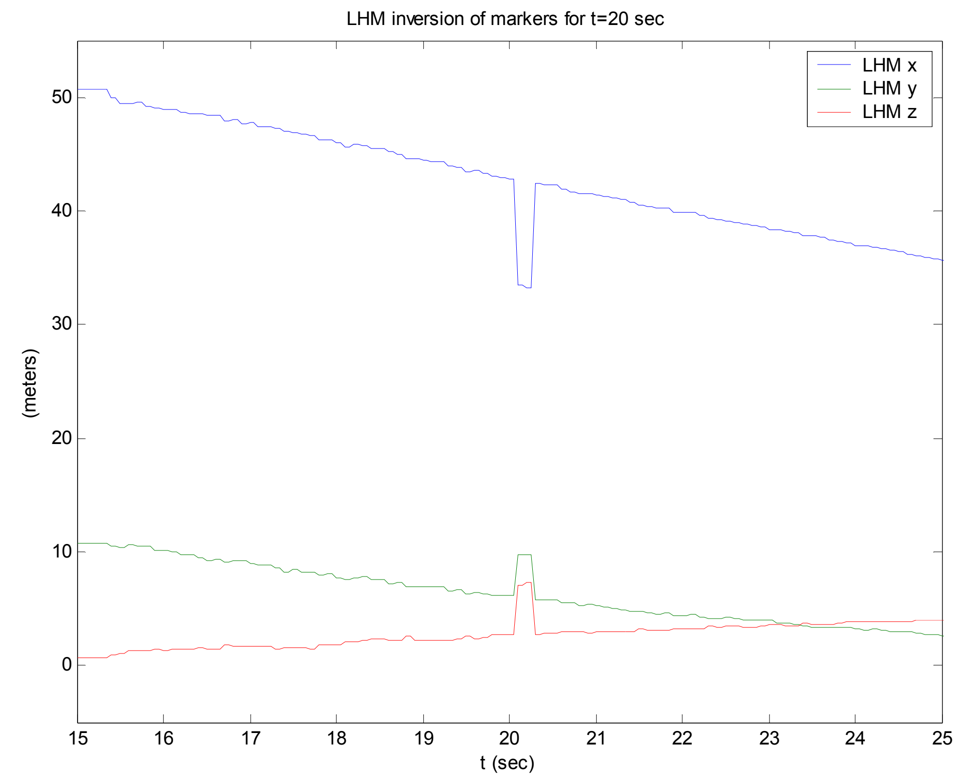


Fig. 7.28: labeling error response with LHM algorithm

7.5.3 A real case: noise addiction in the markers position with uncertain labeling

This test is a simulation of what really happens: the camera gives noisy data to the labeling function. We can have both noisy signals and labeling errors without knowing where in reality the errors are. For this reason we move the noise presented in Fig 7.18 before the labeling function and after the *scale* block. The results are analogous to those obtained in the paragraph 7.5.1 and for a labeling error caused by the noise, the GLSDC algorithm would probably diverge for a noise power of $4*10^{-9}$, and as said previously, once this happens, the GLSDC does not return to the correct estimation. In fact we can observe that the GLSDC algorithm in a real situation is less stable than the LHM algorithm, since the presence of noise and labeling errors is enough to have the divergence of the GLSDC. In Fig 7.29 and 7.30 we can see how the GLSDC behaves in the limit case of a noise power of $3*10^{-9}$, and instead how the LHM algorithm works for a noise power of $4*10^{-9}$. The behavior of LHM algorithm is for some time instant incorrect but when we have a correct labeling and enough seen markers the algorithm comes back to a correct pose estimation.

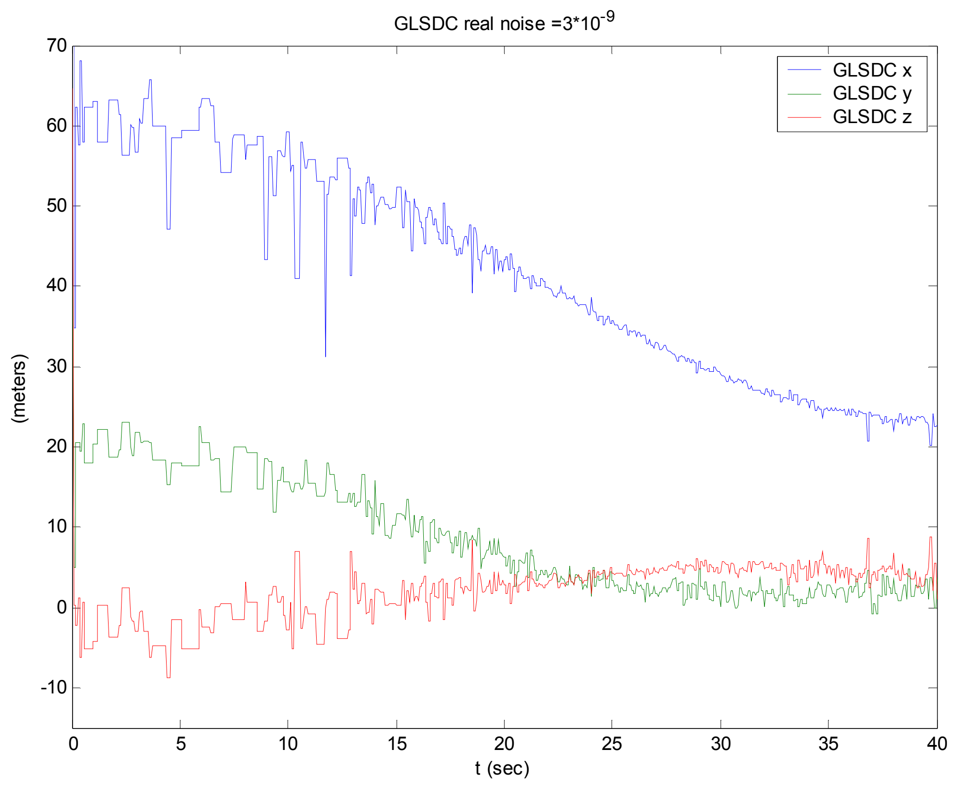


*Fig. 7.29: GLSDC behavior for a noise power of 3*10^{-9} and uncertain labeling*

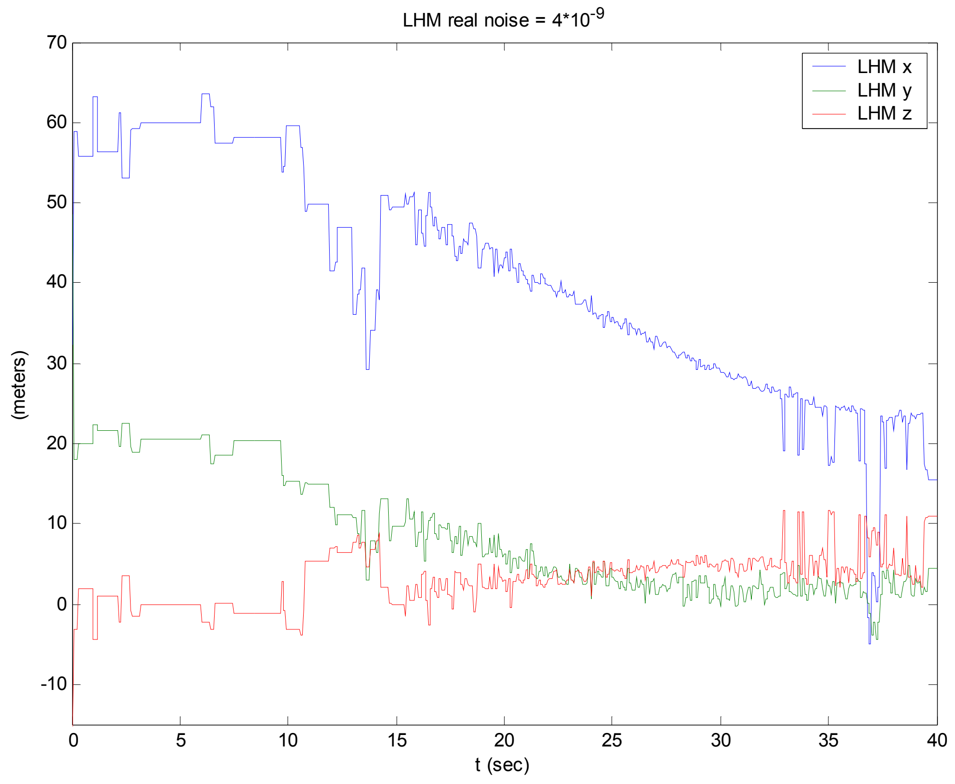


*Fig. 7.30: LHM behavior for a noise power of 4*10^{-9} and uncertain labeling*

7.5.4 Robustness to errors in initial conditions

This test shows the different behaviors of the GLSDC and LHM algorithms with different initial conditions. The initial conditions are composed by the translation vector and the Euler angles. The test is diversified in two parts, the first one consists in the search of a convergence area for the translation vector, for simplicity we varied the vector of one constant quantity for the three components *(x, y, z)*, the second one consist in a search of a convergence area for the yaw angle, these can give an idea of the robustness of the algorithm to errors in the initial condition. We performed the test with a fixed number of markers, specifically the markers {*1, 3, 4, 6, 9*}, surely the convergence area results widened if more markers are used. We consider x_o= [*x y z psi theta phi*] = [60.5 20 -2.5 0 0.467 0] as the exact initial condition, where *x y* and *z* are the relative position expressed in meters between camera and tanker in camera frame and *psi*, *theta* and *phi* are the Euler angles expressed in radiant. The results are presented in Tab. 7.6, and we note immediately that the GLSDC has a limited convergence area while the LHM works in all situations.

Translation vector		**Yaw angle (*psi*)**	
GLSDC interval from exact initial condition	LHM interval from exact initial condition	GLSDC interval from exact initial condition	LHM interval from exact initial condition
[-44.7 65.4]	[-∞ +∞]	[-1.74 3.06] rad = [-100 175] °	[0 2π]

Tab. 7.6: convergence area in the initial condition tests

In Fig. 7.31 we can see how the GLSDC works at the limit of the convergence area for the translation vector, specifically the algorithm shows a settling time of t = 0.4 sec before providing a consistent measure. If we give an initial condition external to the convergence area, the algorithm provides a wrong estimation or diverges according to how much the initial condition is far from the convergence area. We can see in Fig. 7.32 an example of the GLSDC behavior when it starts immediately out of the convergence area. In Fig. 7.33 we show that the LHM algorithm works with initial condition on the translation vector that are considerably distant from the correct initial condition. This is due to the fact that, as seen in chapter 6, the LHM uses an initial condition on the rotation matrix and extracts the translation vector in closed form from the matrix R, therefore the convergence area for this algorithm is unbounded.

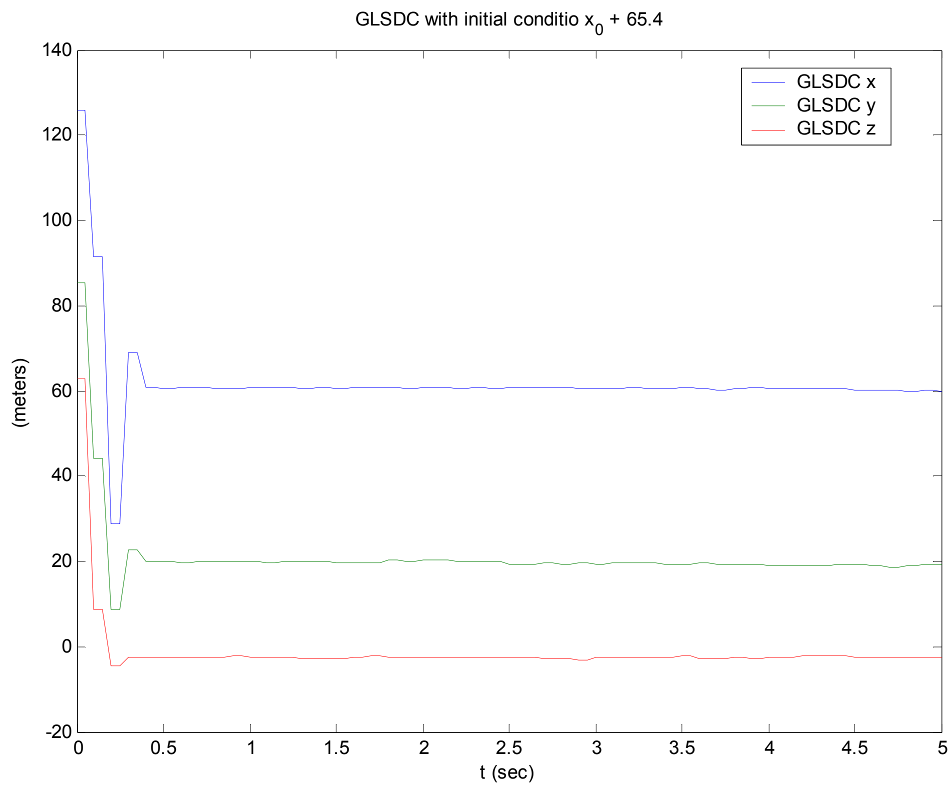


Fig. 7.31: GLSDC algorithm in the limit of the convergence area for the translation vector

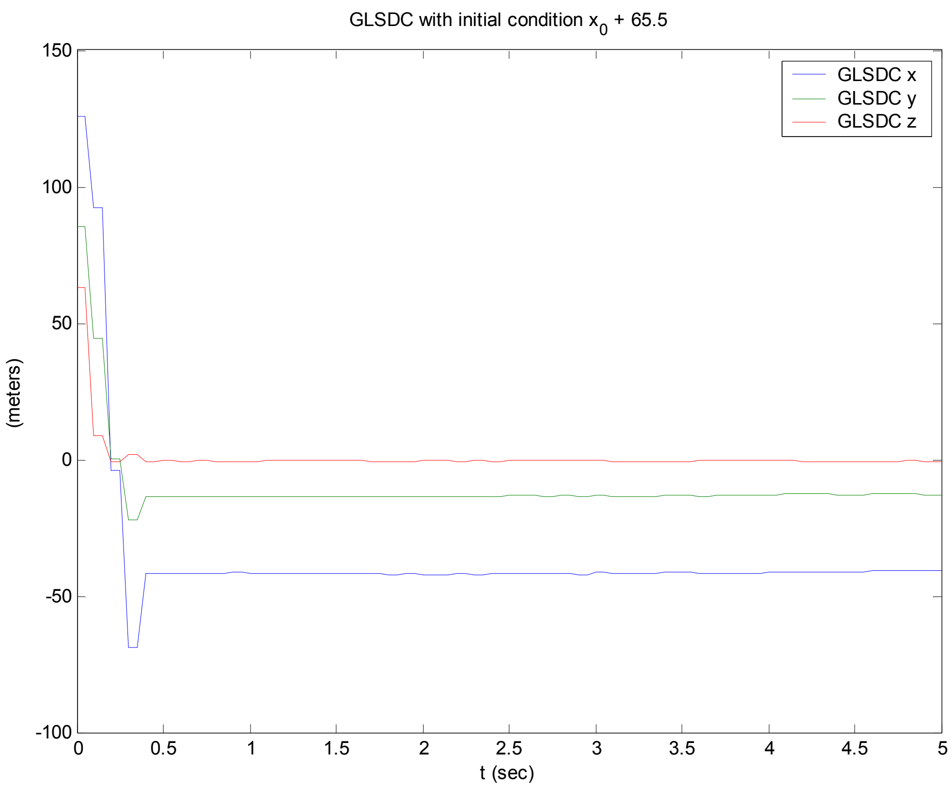


Fig. 7.32: GLSDC algorithm out of the convergence area for the translation vector

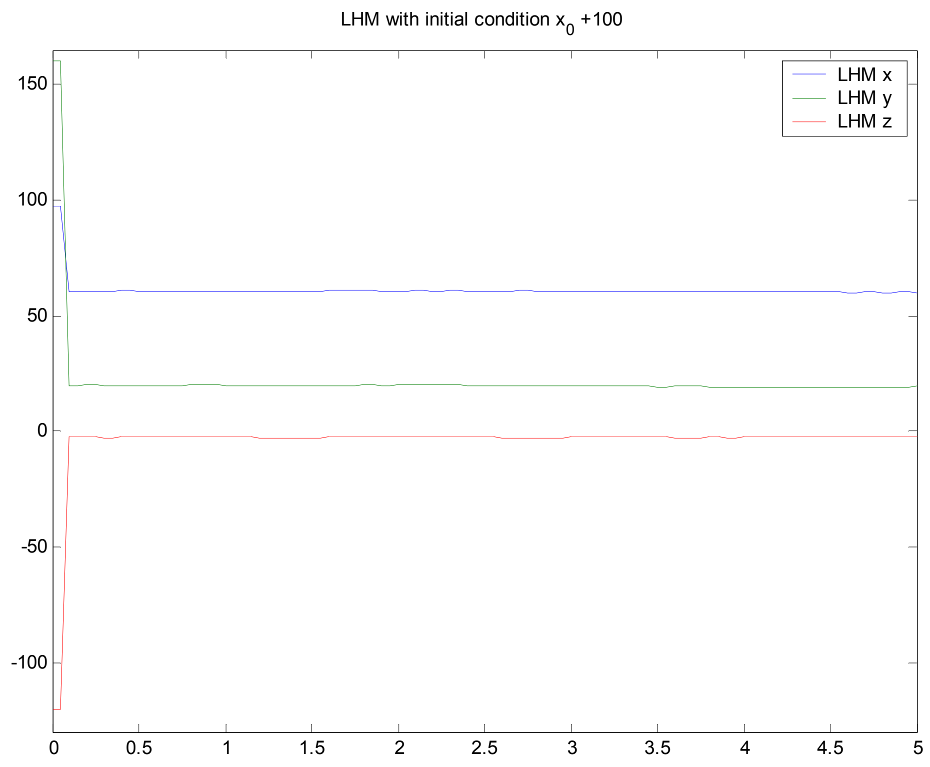


Fig. 7.33: LHM algorithm behavior with initial condition wrong for the translation vector

Now we examine the robustness to errors in the yaw angle. The GLSDC pose estimation algorithm has a limited convergence area even if we change only the yaw angle. In Fig.7.34, and 7.35 we show that a variation on the yaw angle generates a variation on the other angles and this variation cause instability or an incorrect estimation. The LHM algorithm for any given variation on yaw angle does not show problems or incorrect estimation, an example is provided in Fig 7.36. We note that a variation in the yaw angle has influence on the other angles but this influence is bounded on the first execution of the algorithm.

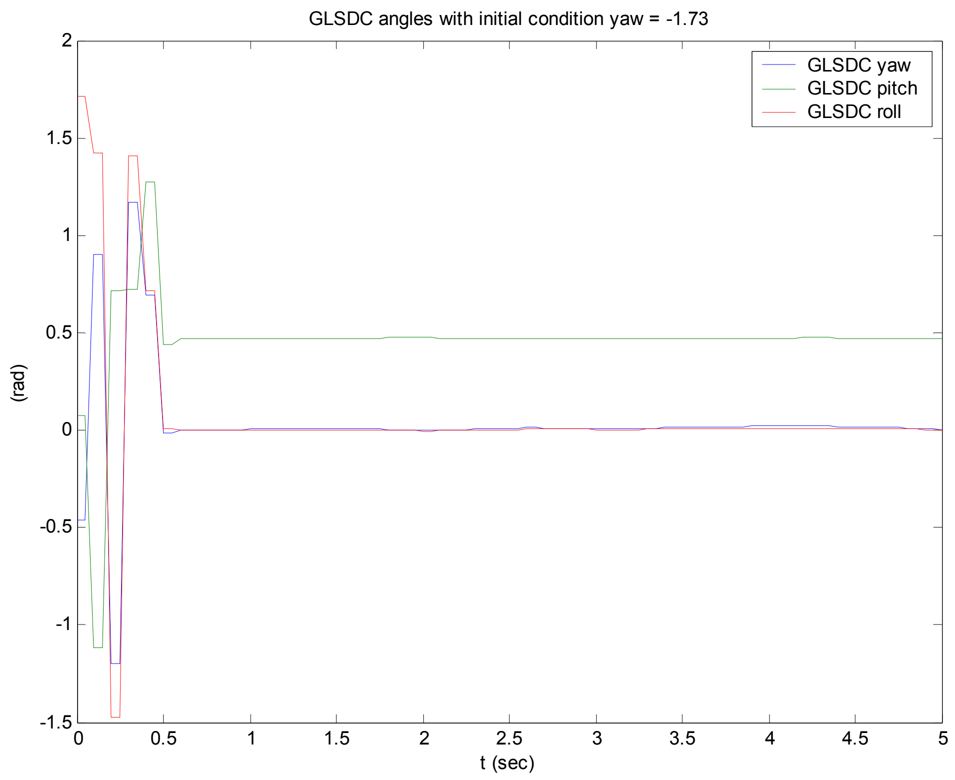


Fig. 7.34: GLSDC algorithm in the limit of convergence area for the yaw angle

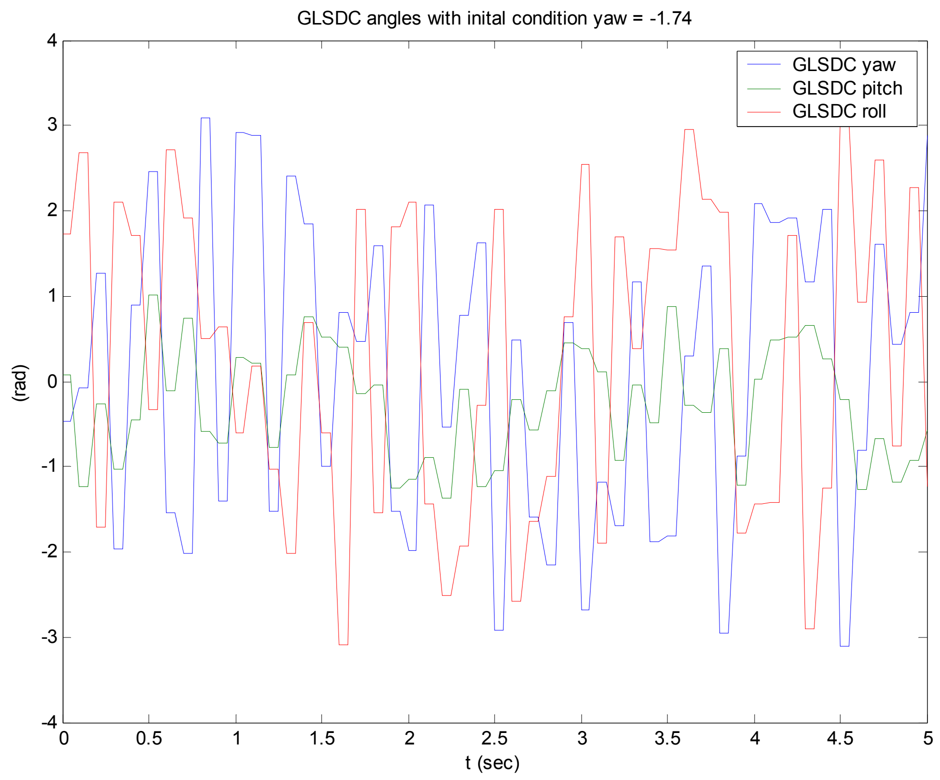


Fig. 7.35: GLSDC algorithm out of the convergence area for the yaw angle (divergence)

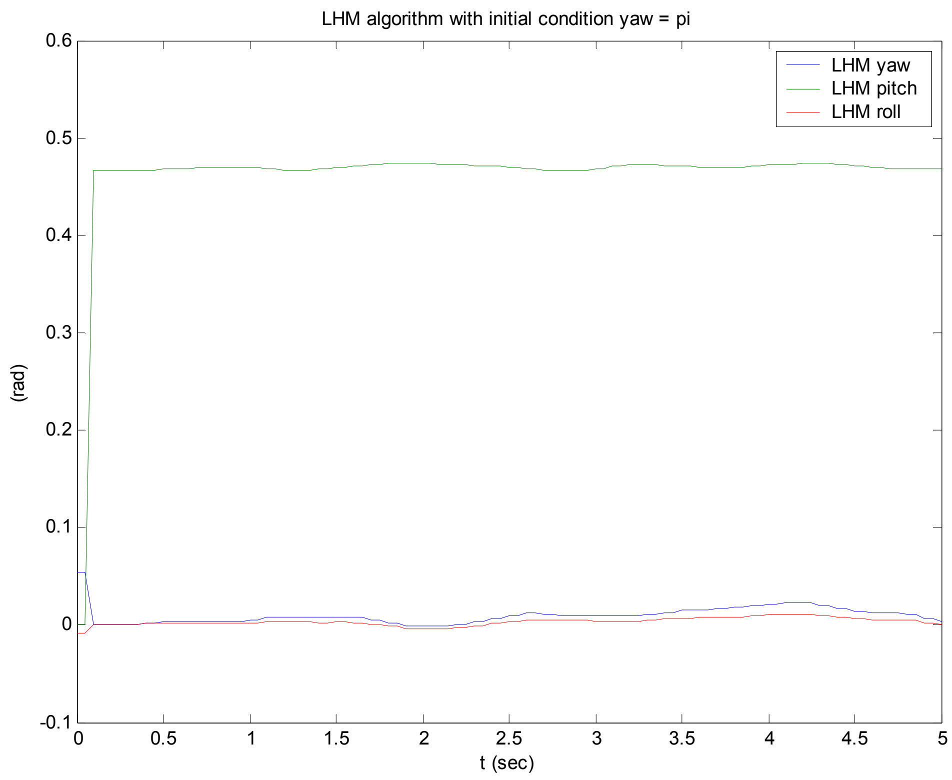


Fig. 7.36: LHM algorithm behavior with initial condition wrong for the yaw angle

7.6 Error propagation analysis

The test that we propose in this section tries to explain how the noises on the position markers influence the estimated position, and how the input noise is propagated to the output. For this test, the position estimation works in SV with seen markers {*1, 3, 4, 6, 9*}. A white Gaussian noise (WGN) with several values of noise power is added on the position *(x, y)* of the marker *1* found by the camera image, and, as a result, the translation vector of the pose estimation algorithm results noisy. We can isolate the output noise by subtracting the non noisy reference output from the noisy output. In the analysis we assume that the all statistic processes are ergodic, which means that time and space distribution averages are equal. In other words, the ergodicity of one process means that the statistics process behavior is the same for a great but limited numbers of samples and for infinite numbers of samples.

The input noise is a white Gaussian noise with mean around 0 and noise power different for each test. We use in the first test a noise power equal to $1*10^{-9}$, the second one has a noise power of $2*10^{-9}$ and the last has a noise power of $3*10^{-9}$. We represent the estimated power spectral density (PSD) of the first noise in Fig 7.37 and the empirical cumulative distribution function (CDF) together with the reference CDF in Fig 7.38. That proves that the input noise is an approximation of WGN. The details of the input noises are presented in Tab 7.7.

	Min	**Max**	**η = Mean**	**σ = STD**
Noise power =1*10^{-9}	-4.856*10^{-4}	4.272*10^{-4}	-1.367*10^{-6}	1.403*10^{-4}
Noise power =2*10^{-9}	-6.867*10^{-4}	6.041*10^{-4}	-1.933*10^{-6}	1.984*10^{-4}
Noise power =3*10^{-9}	8.410*10^{-4}	7.399*10^{-4}	-2.368*10^{-6}	2.431*10^{-4}

Tab. 7.7: input data noise

All the data are sampled with the frequency *f = 10 Hz* as this is the frequency of all the components in the machine vision system. In all the PSD figures we have the normalized frequency on the *x* axis, and the value of 1 (rad /sample) corresponds to the frequency 5 Hz.

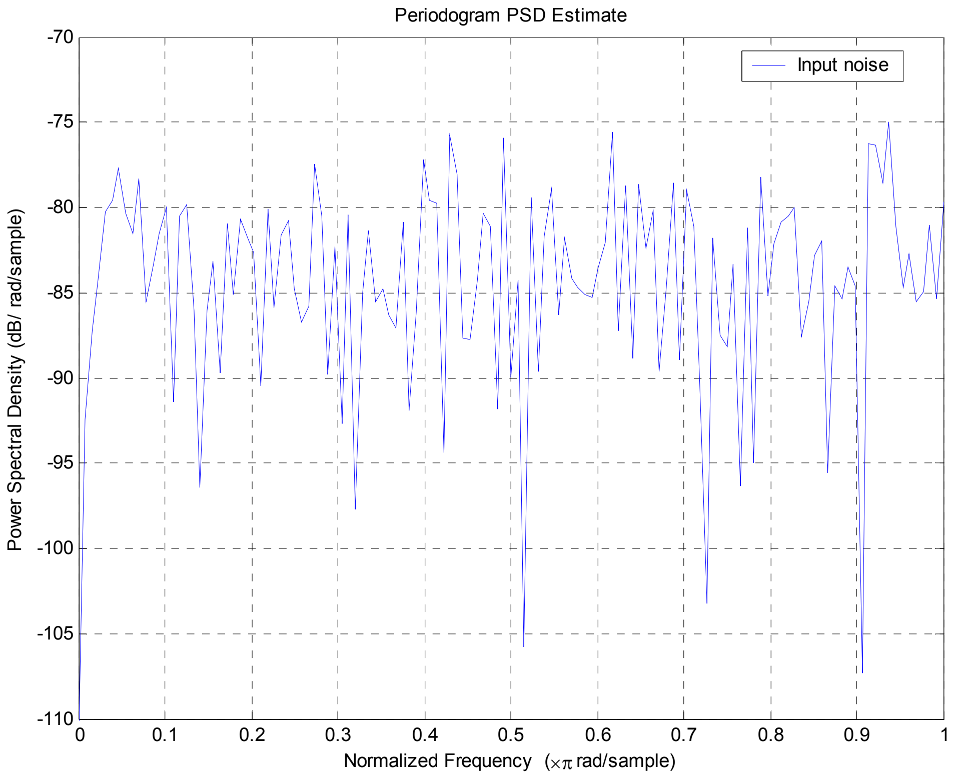


*Fig. 7.37: noise input Power Spectral Density (PSD) for noise power = 1*10^{-9}*

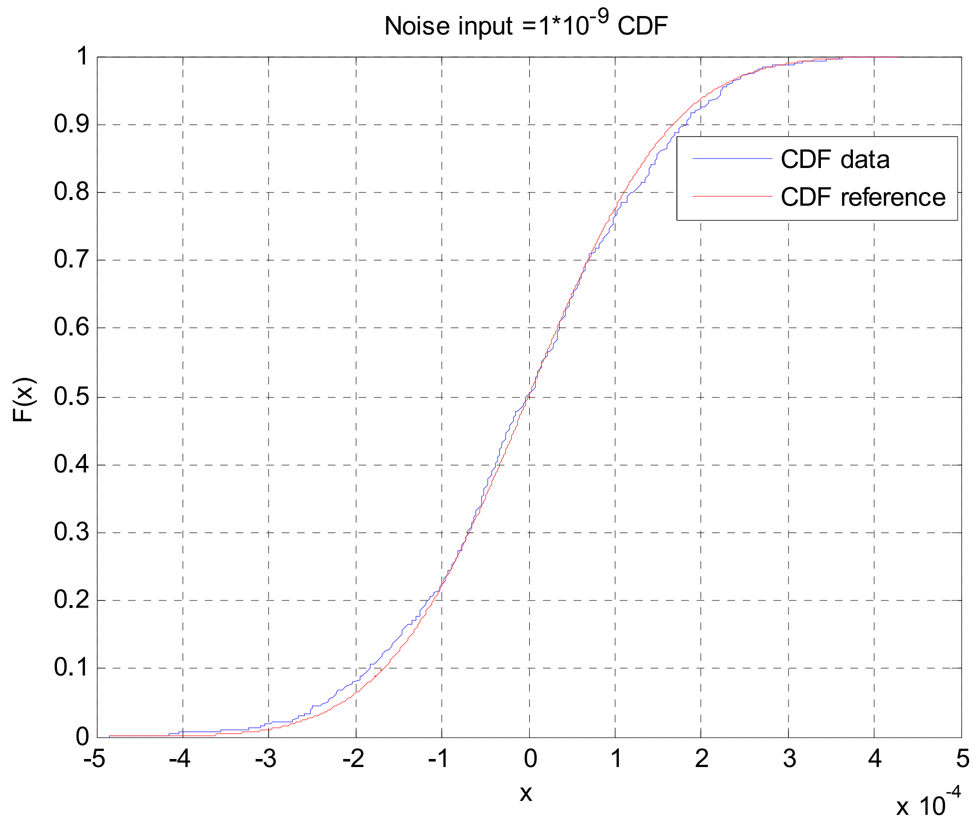


*Fig. 7.38: noise input Cumulative Distribution Function (CDF) for noise power = 1*10^{-9}*

The addition of noise on the position of one marker causes noisy position estimation; we can examine this noise, as aforesaid, by subtracting the data obtained from one non-noisy simulation from the data obtained from a noisy simulation. The PSD of the output noise is characteristic of a white noise for both the algorithms as we can see by Fig. 7.39 and 7.40. According to the theory of stochastic processes, if a Gaussian process goes trough a linear systems the output process is still Gaussian.

At this point it is interesting to investigate in whether the MV system acts as a linear system as far as noise propagation, between an input (on the marker position) and the output(on the translation vector), is concerned. We consider the system composed by one input and three outputs as if they were three systems with one input and one output, and we call this systems $GLSDC_X$, $GLSDC_Y$ and $GLSDC_Z$ for the GLSDC algorithm and LHM_X, LHM_Y and LHM_Z for the LHM algorithm. In addition we define the estimated PSD with the periodogram method:

$$PSD(f) = \frac{2\pi}{n}\left|\sum_{l=1}^{n} x_l e^{-j\omega l}\right|^2 \tag{7.2}$$

where n is the number of element of the noise data and x_l is the position l of the vector.
From the signal theory we know:

$$\frac{PSD_Y(f)}{PSD_X(f)} = \left|H(f)\right|^2 \tag{7.3}$$

where PSD_Y represent the output noise PSD, PSD_X represent the input noise PSD, $H(f)$ is the frequency response of the system.

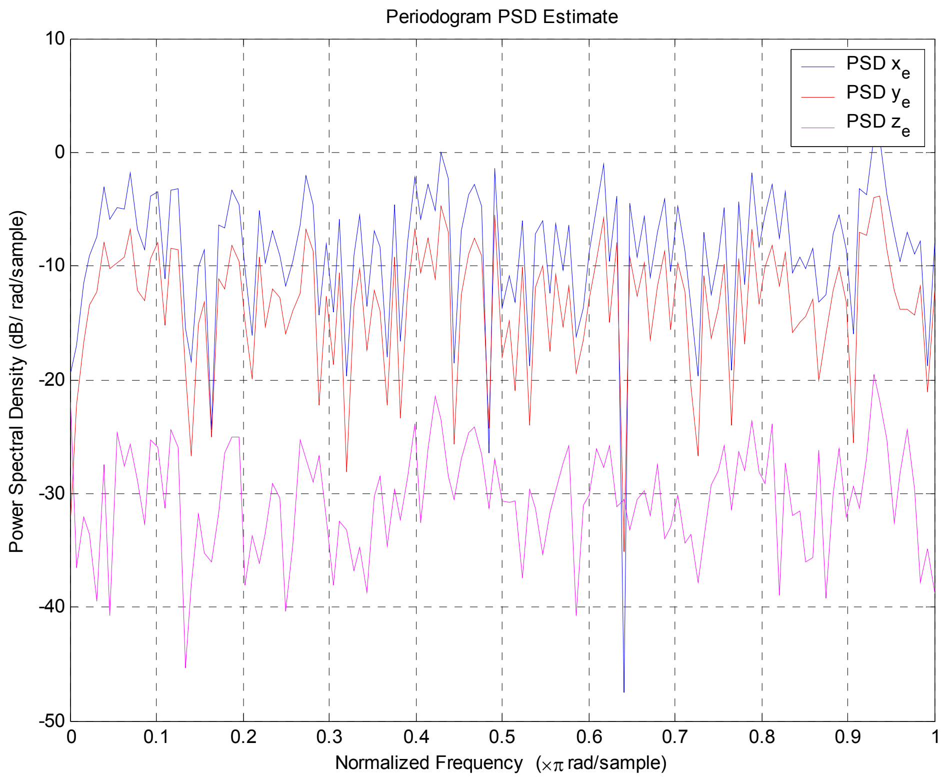


Fig. 7.39: output noises PSD of GLSDC algorithm with noise input power = $1*10^{-9}$

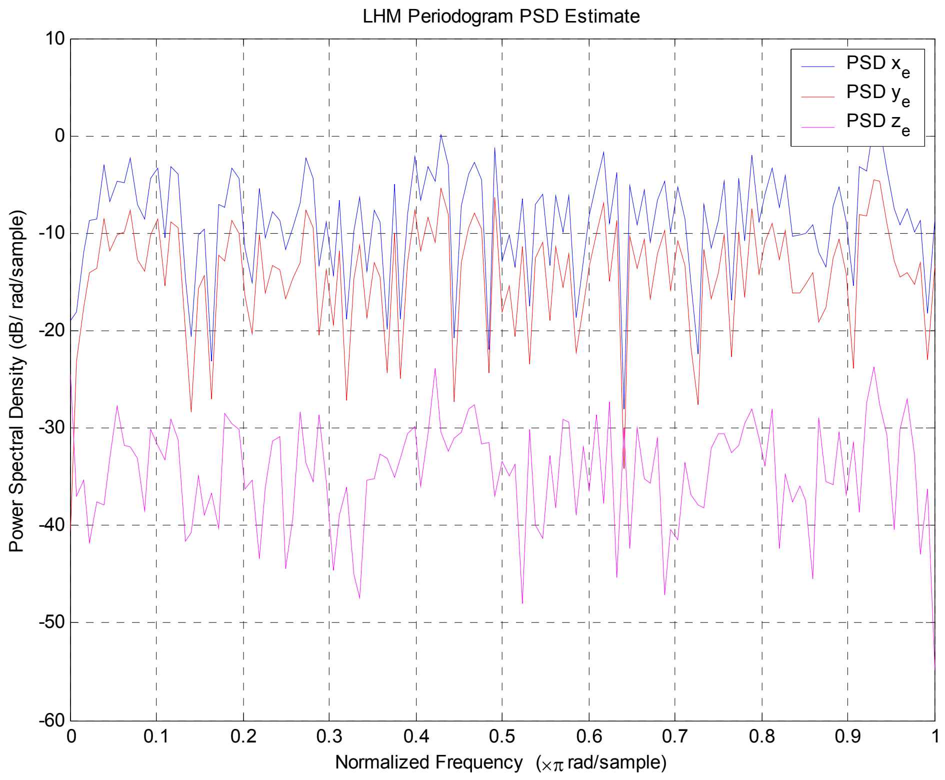


Fig. 7.40: output noises PSD of LHM algorithm with noise input power = $1*10^{-9}$

If the ratio between PSD_Y and PSD_X is almost equal among the various powers of input noise, then for what concerns noise propagation, we can approximate the systems with a linear ones. The linearity is a sufficient condition for the preservation of the Gaussian distribution. We can see in Fig. 7.41, 7.42, 7.43 the square of the frequency response for the systems $GLSDC_X$, $GLSDC_Y$ and $GLSDC_Z$. In Fig 7.44, 7.45 7.46 we present the square of the frequency response of the systems LHM_X, LHM_Y and LHM_Z. It is clearly visible that the behavior of 6 systems is exactly linear, because we have an exact overlap of the line on the 6 figures. Once the linearity is verified, we can establish with safety that the output noise is Gaussian with mean equal to 0 since the input noise has mean value equal to 0 and the following relationship is valid:

$$\eta_Y = H(0)\eta_X \tag{7.4}$$

where η_X is the input mean value, η_Y the output mean value and *H(0)* is the static gain of the system. The output error has variance equal to the second order moment because the error has mean equal to 0, and the second order moment is provided by the relationship:

$$E\{X^2(t)\} = 2\int_0^{\infty} PSD_Y(f)df \tag{7.5}$$

where PSD_Y is the power spectral density of the output noise.

We now have all the information to model the output noise. We are sure that if the input noise is WGN, the output noise will be WGN and moreover we know that the systems behaves as a linear system as far as noise transmission is of concern.

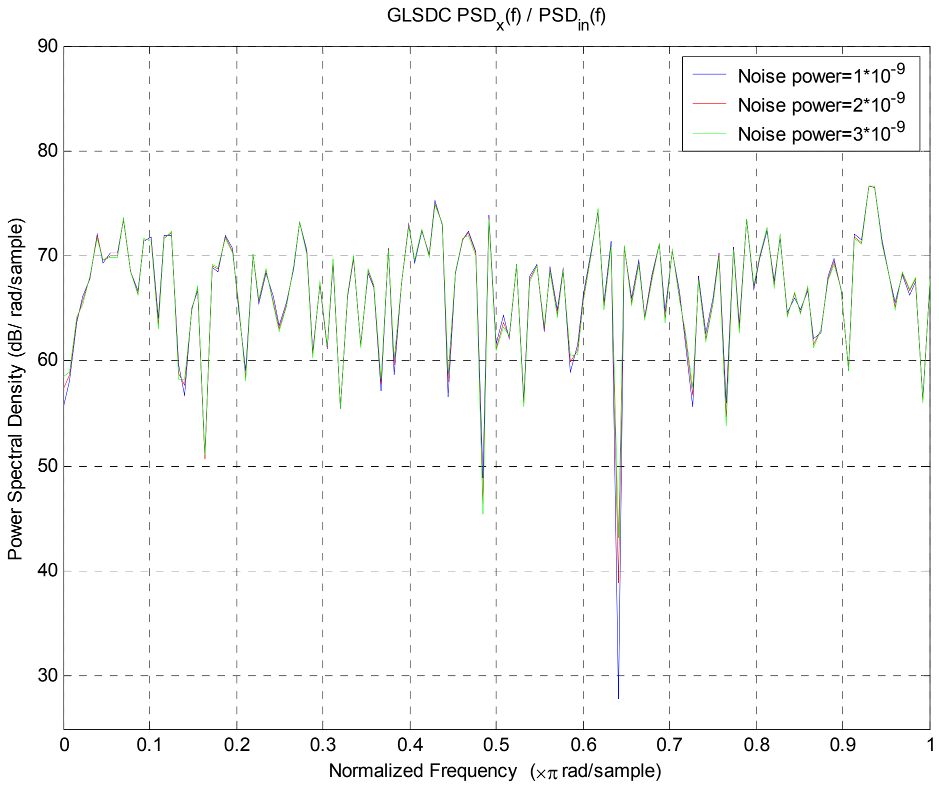


Fig. 7.41: verification of linearity propriety for $GLSDC_X$ system

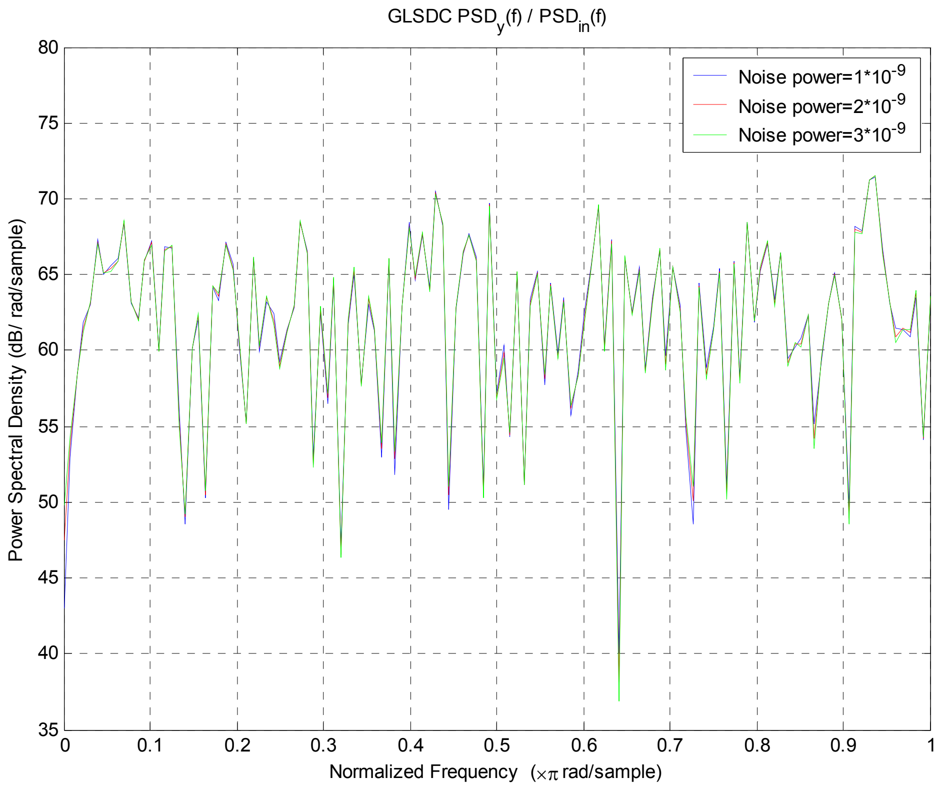


Fig. 7.42: verification of linearity propriety for GLSDC$_Y$ system

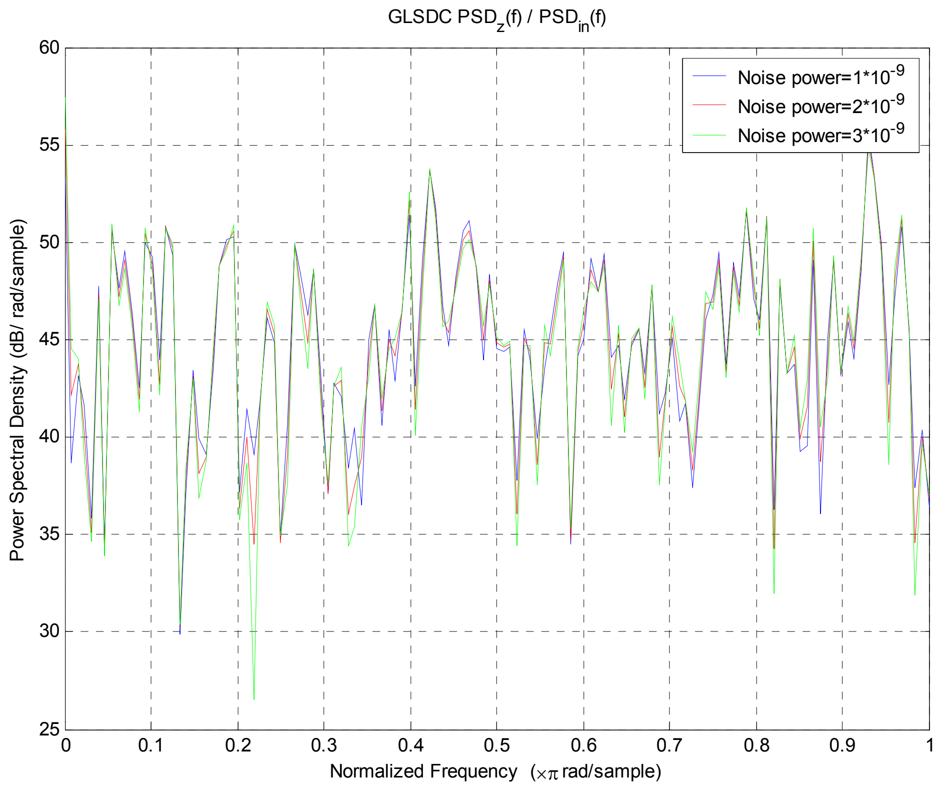


Fig. 7.43: verification of linearity propriety for GLSDC$_Z$ system

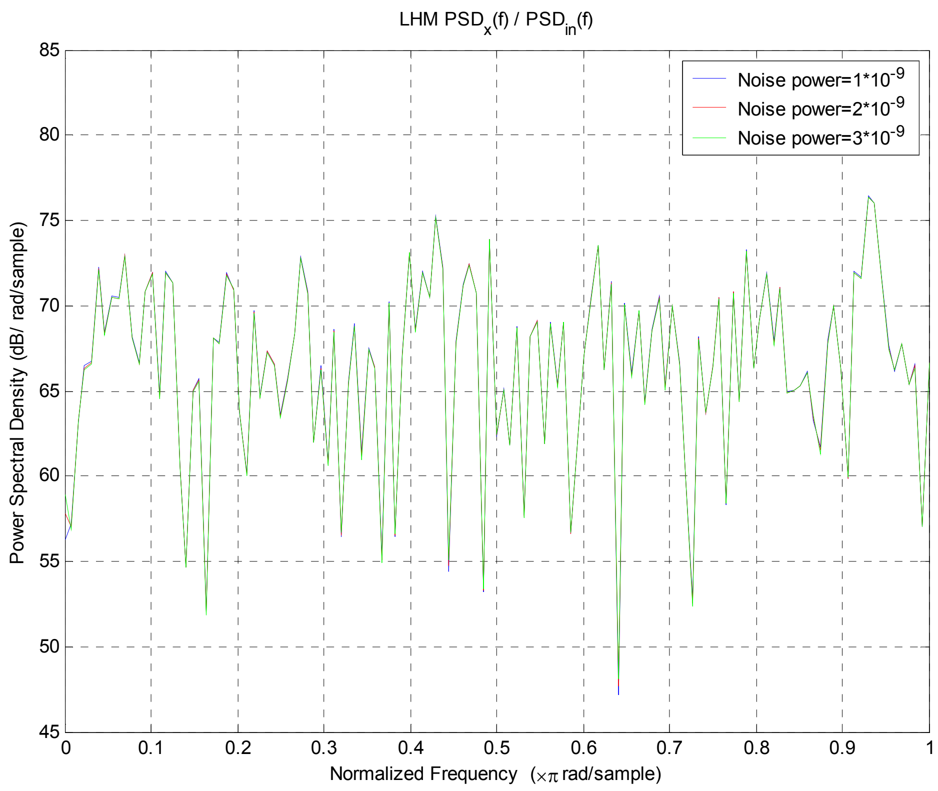


Fig. 7.44: verification of linearity propriety for LHM$_X$ system

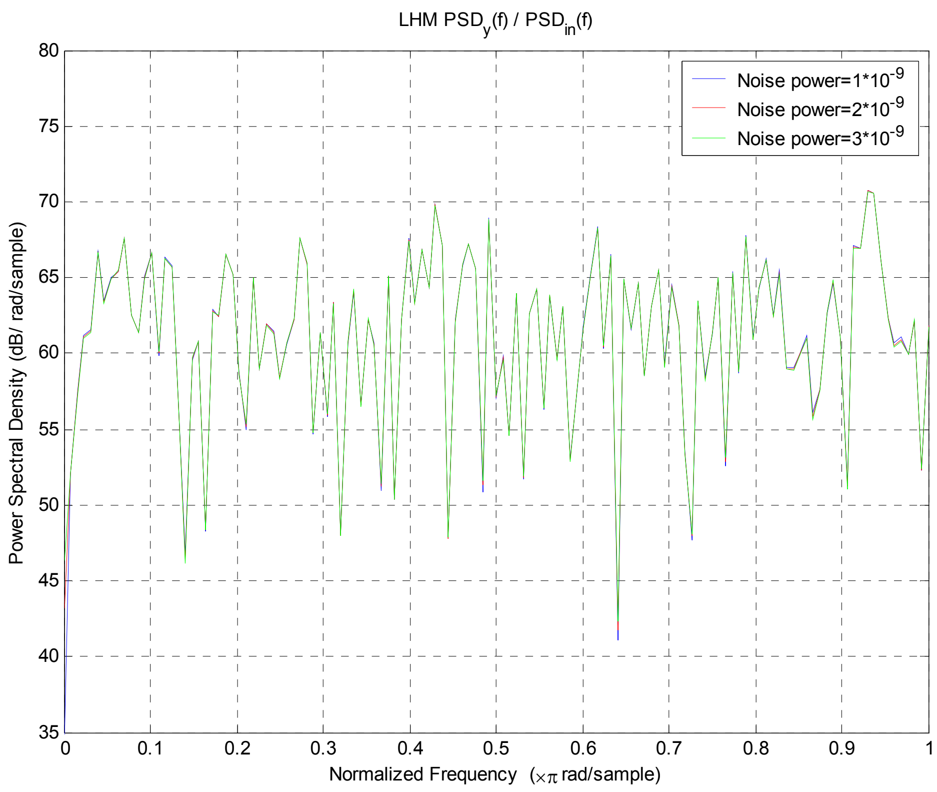


Fig. 7.45: verification of linearity propriety for LHM$_Y$ system

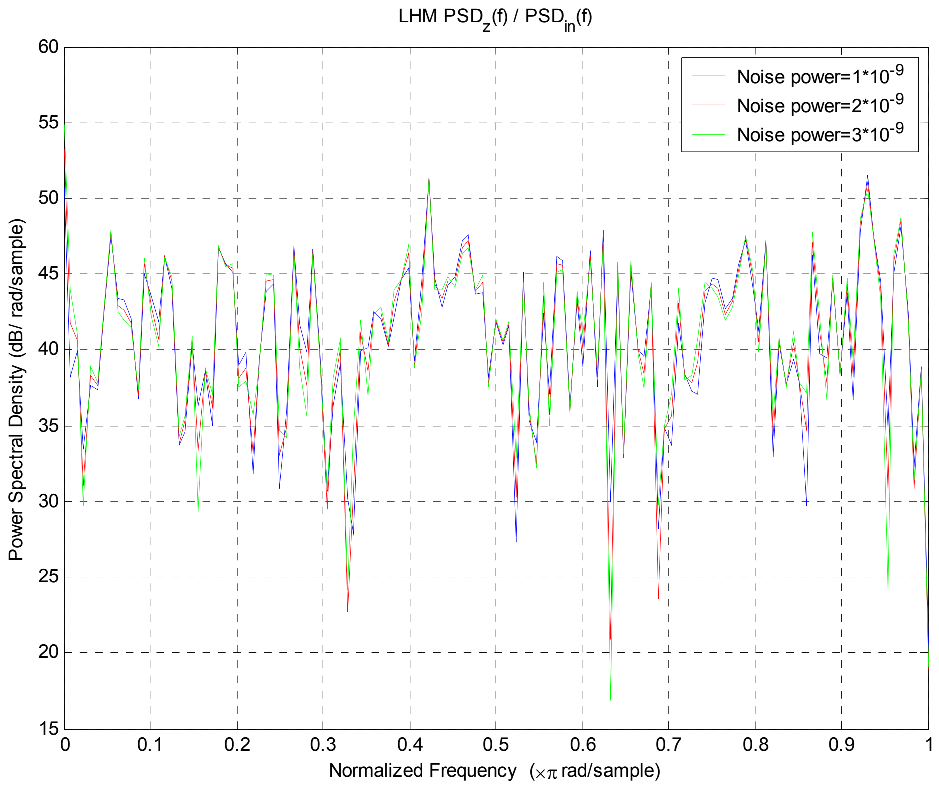


Fig. 7.46: verification of linearity propriety for LHM$_Z$ system

At this point we have to compare the output noise between the three systems of GLSDC and LHM, to understand which algorithm amplifies the noise. In Fig 7.47 -7.49, a direct comparison between the PSD of the systems GLSDC$_X$ and LHM$_X$, GLSDC$_Y$ and LHM$_Y$, GLSDC$_Z$ and LHM$_Z$ are shown. In Fig 7.47 and 7.48 the lines are overlapped which means that the GLSDC and LHM algorithms propagate the errors in the same way. In Fig 7.49, the GLSDC algorithm amplifies the noise more than LHM algorithm as it concerns the variable *z*.

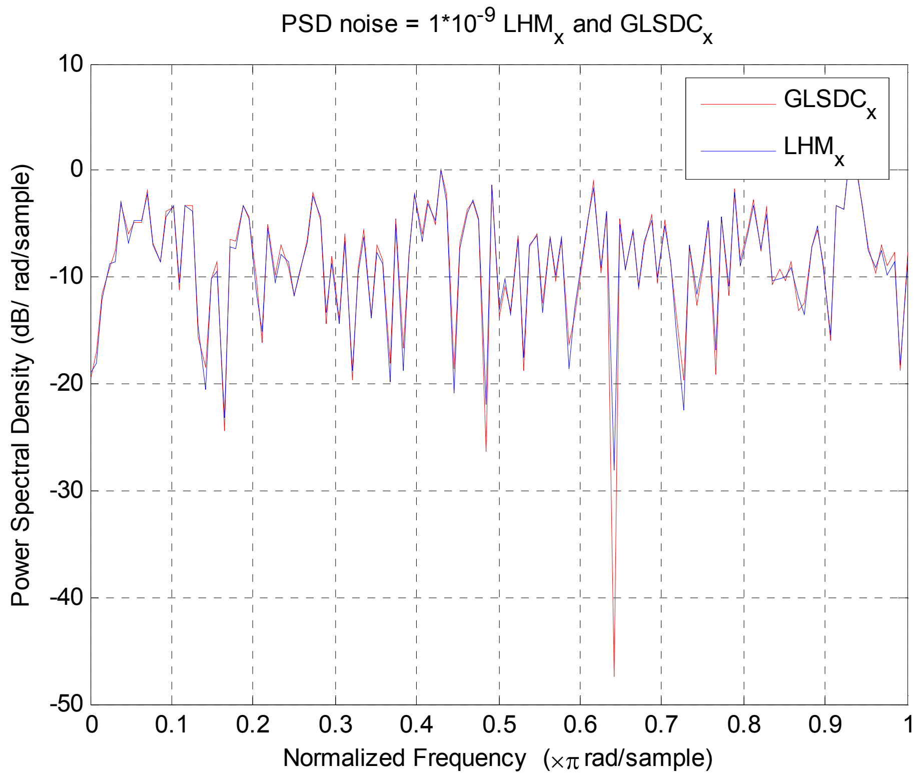


*Fig. 7.47: PSD of GLSDC$_X$ and LHM$_X$ with noise 1*10^{-9}*

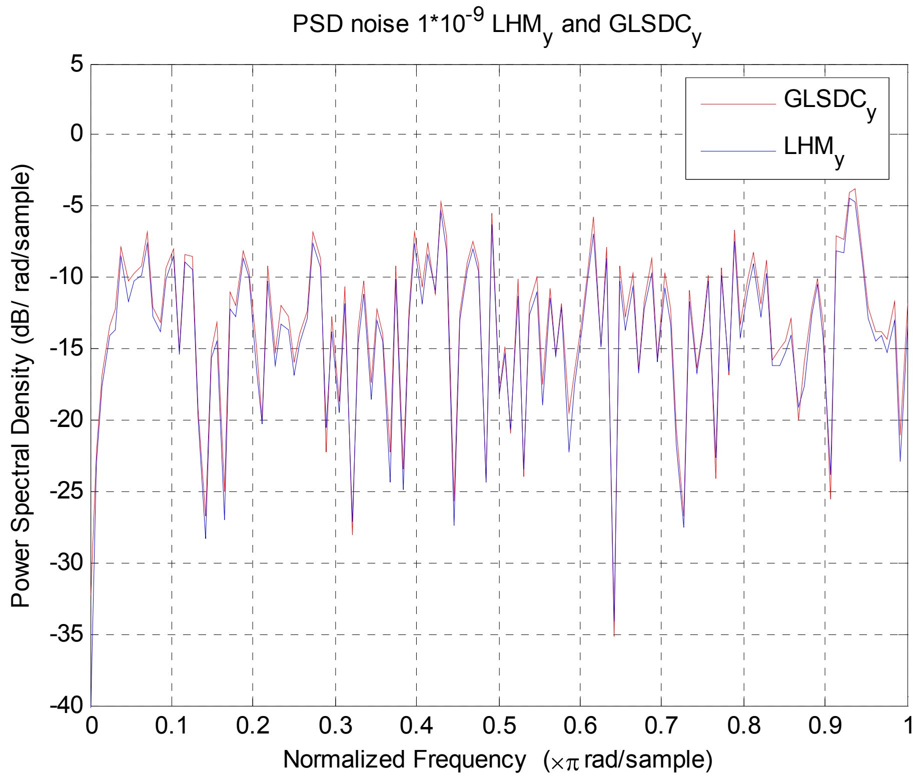


*Fig. 7.48: PSD of GLSDC$_Y$ and LHM$_Y$ with noise 1*10^{-9}*

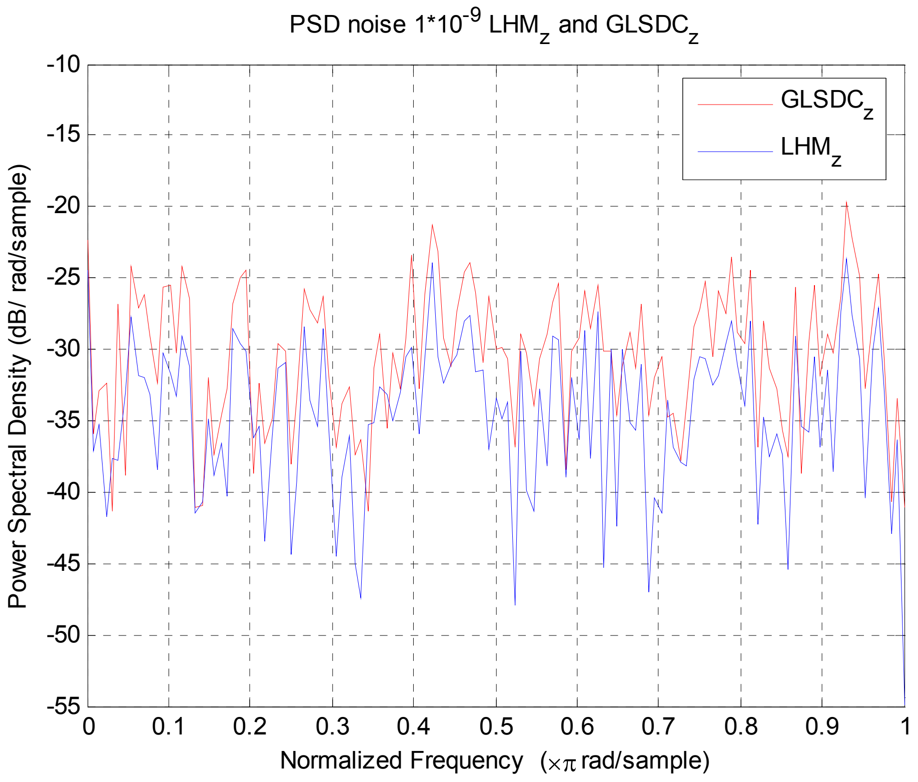


Fig. 7.49: PSD of $GLSDC_Z$ and LHM_Z with noise $1*10^{-9}$

8. CONCLUSIONS

This thesis described different MV algorithms that were developed and jointly tested within a simulation environment specifically developed for the study of the MV-based Autonomous Aerial Refueling problem. In particular, the attention focused on the development of an accurate labeling algorithm that avoids typical errors and on the analysis of the performance of the two most used pose estimation algorithms - the GLSDC and the LHM algorithms - in terms of speed, accuracy, robustness and errors propagation. The results from this detailed comparison indicate that the accuracy of the two algorithms is substantially similar; however, the LHM algorithm provides a substantial higher level of robustness at the expense of a larger required computational effort.

9. REFERENCE

[1] M.L. Fravolini, A. Ficola, M.R. Napolitano G. Campa, M.G. Perhinschi, “Development of Modelling and Control Tools for Aerial Refueling for UAV’s”, Proceedings of the 2003 AIAA GNC Conference, Paper 2003-5798, Austin (TX), August 2003.

[2] L. Pollini, G. Campa, F.Giulietti, M.Innocenti, “Virtual Simulation Set-up for UAVs Aerial Refueling” , Proceedings of the 2003 AIAA Conference on Modeling and Simulation Technologies and Exhibits, Paper 2003-568211-14, Austin (TX), August 2003.

[3] R.M. Harlalick et al., “Pose Estimation from Corresponding Point Data”, IEEE Trans. Systems, Man, and Cybernetics, vol. 19, no. 6, pp. 1,426-1,446, 1989.

[4] C-P Lu, G. Hager, and E. Mjolsness, "Fast and Globally Convergent Pose Estimation From Video Images", IEEE Transaction on Pattern analysis and machine intelligence, vol. 22, pp. 610 - 622, 2000.

[5] A. Ansar and K. Daniilidis, "Linear pose estimation form point or lines", IEEE Transaction on Pattern analysis and machine intelligence, vol. 25, no. 4, April 2003.

[6] O. Faugeras, Three-Dimensional Computer Vision. MIT Press, 1993.

[7] B.K.P. Horn, “Closed-Form Solution of Absolute Orientation Using Unit Quaternion”, J. Optical Soc. Am., vol. 4, pp. 629-642, 1987.

[8] M.W. Walker, L. Shao, and R.A. Volz, “Estimating 3D Location Parameters Using Dual Number Quaternions”, CVGIP: Image Understanding, vol. 54, no. 3, pp. 358-367, 1991.

[9] K.S. Arun, T.S. Huang, and S.D. Blostein, “Least-Squares Fitting of Two 3D Point Sets”, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 9, pp. 698-700, 1987.

[10] M. Ciampi, L. Verazzani, G. Del Corso, “Teoria dei segnali, Segnali aleatori”, ETS Pisa, 1984.

[11] A. Caiti, “Appunti di Identificazione dei Sistemi Incerti”, Università di Pisa, 2004.

[12] B.K.P. Horn, H.M. Hilden, and S. Negahdaripour, “Closed-Form Solution of Absolute Orientation Using Orthonomal Matrices”, J.Optical Soc. Am., vol. 5, pp. 1,127-1,135, 1988.

[13] D.G. Luenberger, Linear and Nonlinear Programming. second ed. Reading, Mass.: Addison Wesley, 1984.

[14] W. Szczepanski. “Die LBsungsvorschlage fur den raumlichen Ruckwartheinschnitt. Deutsche Geodatische Kommission.” Reihe C:Dissertationcn-Heft Nr. 29, pp. 1-144, 1958.

[15] M. A. Fischler and R. S. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography.” C*ommunications of the ACM,* vol. 24, no. 6, June 1981.

[16] B. Wrobel and D. Klemm. “Uber die Vermeidung singularer Falle bel der Berechnung allgemeiner raumlicher Drehungen.” in *Proc. XVrh Congress of ISPRS.* Rio de Janeiro, Brazil, 1984. and in the *Int. Archives of Photogrammetry and Remote Sensing,* vol. 25, part A3b. pp. 1153-1163.

[17] E. H. Thompmn. “An exact linear solution of the problem of absolute orientation.” Phorogrunimerriu, vol. 15, no. 4. pp. 163-178. 1958.

[18] G . H. Schut, “On exact linear equations for the computation of rotational element of absolute orientation.” *Photogrammetria,* vol.15, no. 1. pp. 34-37, 1960.

[19] J. M. Tienstra. “Calculation of orthogonal matrices,” ITC DelftSeries. A4X. 1969.

[20] J. A. Pope, “An advantageous. alternative parametrization of rotations for analytical photogrammetry,” ESSA Tech. Rep., C and GS 39. 1970.

[21] J. A. R. Blais, “Three-dimensional similarity,” *The Canadian Surveyor .* no. 1, 1972. pp. 71-76.

[22] F. Sanso. “An exact solution of the roto-translation problem.” *Photogrammetria.* vol. 29. pp. 203-216, 1973.

[23] K. S. Arun, T. S. Huang, and S. D. Rlostein. "Least-squares fitting of two 3-D point sets," *IEEE Trans. Pattern Anal . Machine Intell.,* vol. PAMI-9, no. *5.* pp. 698-700. Sept. 1987.
[24] R.M. Haralick. H. Joo, C. Lee. X. Zhuang, V. Vaidya. and M. Kim. "Pose estimation from corresponding point data," *IEEE Computer Society Workshop on Computer Vision,* Miami Beach, FL. Nov. 30-Dec. 3, 1987. pp. 258-263.
[25] R.M. Haralick and L.G. Shapiro, *"Computer and Robot Vision*". Reading, Mass.: Addison-Wesley, 1993.
[26] D.G. Lowe, "Three-Dimensional Object Recognition from Single Two-Dimensional Image", *Artificial Intelligence*, vol. 31, 1987, pp. 355- 395.
[27] Kimmett J., Valasek J., Junkins J.L., "Autonomous Aerial Refueling Utilizing a Vision Based Navigation System", Proceedings of the 2002 AIAA GNC Conference, Paper 2002-5569, Monterey (CA), August 2002.
[28] A. Fusiello "Visione Computazionale: Appunti delle lezioni" Università di Verona, 2003-2004
[29] S. Hutchinson, G. Hager, P. Corke, "A tutorial on visual servo control", *IEEE Trans. On Robotics and Automation*, vol. 12, no. 5, 1996, pp. 651-670.
[30] Lowe, D. G., "Fitting parameterized three-dimensional models to images," *IEEE Transactions on Pattern Analysis and Machine intelligence*, vol. 13, no. 5, 1991, pp. 441-450.
[31] S. Vogt, A. Khamene, F. Sauer, H. Niemann, "Single Camera Tracking of Marker Clusters: Multiparameter Cluster Optimization and Experimental Verification", Proceedings of International Symposium on Mixed and Augmented Reality, Darmstadt, Germany , pp. 127–136, Sept. 2002.
[32] B.L. Stevens, F.L. Lewis, *"Aircraft Control and Simulation",* John Wiley & Sons, New York, 1987.
[33] Addington, G.A., Myatt, J.H., "Control-Surface Deflection Effects on the Innovative Control Effectors (ICE 101) Design, "Air Force Report", AFRL-VA-WP-TR-2000-3027, June 2000.
[34] Roskam J. "*Airplane Flight Dynamics and Automatic Flight Controls – Part II*", DARC Corporation, Lawrence, KS, 1994.
[35] Blake W, Gingras D.R., "Comparison of Predicted and Measured Formation Flight Interference Effect", Proceedings of the 2001 AIAA Atmospheric Flight Mechanics Conference, AIAA Paper 2001-4136, Montreal, August 2001.
[36] Gingras D.R., Player J.L., Blake W., "Static and Dynamic Wind Tunnel testing of Air Vehicles in Close Proximity", Proceedings of the 2001 AIAA Atmospheric Flight Mechanics Conference, Paper 2001-4137, Montreal, Canada, August 2001.
[37] Philip N.K., Ananthasayanam M.R., "Relative Position and Attitude Estimation and Control Schemes for the Final Phase of an Autonomous Docking Mission of Spacecraft", *Acta Astronautica*, vol. 52, 2003, pp. 511-522.
[38] Sinopoli B., Micheli M., Donato G., Koo T.J., "Vision Based Navigation for an Unmanned Aerial Vehicle", Proceedings of the 2001 IEEE International Conference on Robotics and Automation, Vol. 2, 1757-1764, Seoul, South Korea, May 2001.
[39] Addington, G.A., Myatt, J.H., "Control-Surface Deflection Effects on the Innovative Control Effectors (ICE 101) Design,"Air Force Report, AFRL-VA-WP-TR-2000-3027, June 2000.
[40] M.L. Fravolini, G. Campa, A. Ficola, M.R. Napolitano, B. Seanor, "Modeling and Control issues for Machine Vision based Autonomous Aerial Refueling", Journal of Guidance Control and dynamics, In Press.
[41] M.L. Fravolini, A. Ficola, G. Campa, M.R. Napolitano, B. Seanor, "Modeling and Control issues for Autonomous Aerial Refueling for UAVs using a Probe-Drogue refueling system", Aerospace science of technology, Vol. 8, no. 7, pp. 611-618. Oct. 2004.

[42] G. Campa, M.L. Fravolini, A. Ficola, M.R. Napolitano, B. Seanor, M.G. Perhinschi, "Autonomous aerial refueling for UAVs using a combined GPS-machine vision guidance", AIAA Guidance, Navigation, and Control Conference and Exhibit; Providence, pp. 1-11. 2004, 16-19 Aug. 2004, RI, USA.

[43] L. Pollini, R. Mati and M. Innocenti, G. Campa and M. Napolitano, "A Synthetic Environment for Simulation of Vision-Based Formation Flight", AIAA Modeling and Simulation Technologies Conference and Exhibit, Austin, Texas, Aug. 11-14, 2003.

[44] L. Pollini, R. Mati and M. Innocenti, "Experimental evaluation of vision algorithms for formation flight and aerial refueling", American institute of Aeronautics and Astronautics, In Press.

[45] Umeyama, S., "Parameterized point pattern matching and its application to recognition of object families," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol.15, No.2, 1993, pp.136-144.

[46] Pla, F., Marchant, J.A., "Matching Feature Points in Image Sequences through a Region-Based Method," *Computer vision and image understanding*, Vol. 66, No. 3, 1997, pp. 271-285.

[47] Fravolini, M.L., Campa, G., Napolitano, M.R., Ficola, A., "Evaluation of Machine Vision Algorithms for Autonomous Aerial Refueling for Unmanned Aerial Vehicles", Submitted to: *AIAA Journal of Aerospace Computing, Information and Communication,* April 2005.

10. APPENDIX

10.1 Homogeneous Coordinate

Homogenous coordinates utilize a mathematical trick to embed three-dimensional coordinates and transformations into a four-dimensional matrix format. As a result, inversions or combinations of linear transformations are simplified to inversion or multiplication of the corresponding matrices. Homogenous coordinates also make it possible to define perspective transformations.

10.1.1 4x1 Homogeneous coordinate vector

Instead of representing each point (x,y,z) in three-dimensional space with a single three-dimensional vector:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{10.1}$$

homogenous coordinates allow each point (x,y,z) to be represented by any of an infinite number of four dimensional vectors:

$$\begin{bmatrix} T*x \\ T*y \\ T*z \\ T \end{bmatrix} \tag{10.2}$$

The three-dimensional vector corresponding to any four-dimensional vector can be computed by dividing the first three elements by the fourth, and a four-dimensional vector corresponding to any three-dimensional vector can be created by simply adding a fourth element and setting it equal to one.

Many textbooks define homogenous coordinates in such a way that points are represented by 1x4 vectors:

$$\begin{bmatrix} T*x & T*y & T*z & T \end{bmatrix} \tag{10.3}$$

instead of 4x1 vectors. This definition is not used in the AIR package and results in different 4x4 homogenous coordinate transformation matrices than those described below.

10.1.2 4x4 Homogenous Coordinate Transformation Matrices

Homogenous coordinate transformation matrices operate on four-dimensional homogenous coordinate vector representations of traditional three-dimensional coordinate locations. Any three-dimensional linear transformation (rotation, translation, skew, perspective distortion) can be represented by a 4x4 homogenous coordinate transformation matrix. In fact, because of the redundant representation of three space in a homogenous coordinate system, an infinite number of different 4x4 homogenous coordinate transformation matrices are available to perform any given linear transformation. This redundancy can be eliminated to provide a unique representation by dividing all elements of a 4x4 homogenous transformation matrix by the last element (which will become equal to one). This means that a 4x4 homogenous transformation matrix can incorporate as many as 15 independent parameters. The generic format representation of a homogenous transformation equation for mapping the three dimensional coordinate (x,y,z) to the three-dimensional coordinate (x',y',z') is:

$$\begin{bmatrix} T'*x' \\ T'*y' \\ T'*z' \\ T' \end{bmatrix} = \begin{bmatrix} T''*a & T''*b & T''*c & T''*d \\ T''*e & T''*f & T''*g & T''*h \\ T''*i & T''*j & T''*k & T''*m \\ T''*n & T''*p & T''*q & T'' \end{bmatrix} \begin{bmatrix} T*x \\ T*y \\ T*z \\ T \end{bmatrix} \tag{10.4}$$

If any two matrices or vectors of this equation are known, the third matrix (or vector) can be computed and then the redundant T element in the solution can be eliminated by dividing all elements of the matrix by the last element.

Various transformation models can be used to constrain the form of the matrix to transformations with fewer degrees of freedom.

In many textbooks, you will find homogenous transformation matrices defined such that 1x4 homogenous coordinate vectors are placed to the left of the 4x4 homogenous coordinate transformation matrix and multiplied.

10.1.3 Translations

Translations can be represented by the 4x4 homogenous coordinate transformation matrix:

$$\begin{bmatrix} 1 & 0 & 0 & x-shift \\ 0 & 1 & 0 & y-shift \\ 0 & 0 & 1 & z-shift \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.5}$$

where:

$x-shift$ = translation along the x axis

$y-shift$ = translation along the y axis

$z-shift$ = translation along the z axis

10.1.4 Rotation

A series of rotations (in the order [roll matrix]*[pitch matrix]*[yaw matrix]) can be represented by the 4x4 homogenous coordinate transformation matrix:

$$\begin{bmatrix} \cos\theta\cos\psi & -\cos\theta\sin\psi+\cos\psi\sin\theta\sin\varphi & \sin\psi\sin\varphi+\cos\psi\sin\theta\cos\varphi & 0 \\ \cos\theta\sin\psi & \cos\psi\cos\varphi+\sin\theta\sin\psi\sin\varphi & -\cos\psi\sin\varphi+\cos\varphi\sin\psi\sin\theta & 0 \\ -\sin\theta & \cos\theta\sin\varphi & \cos\theta\cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.6}$$

where

φ = rotation around the x axis (roll)

θ = rotation around the y axis (pitch)

ψ = rotation around the z axis (yaw)

10.1.5 Rescaling

Rescaling along the major axes can be represented by the 4x4 homogenous coordinate transformation matrix:

$$\begin{bmatrix} x-scale & 0 & 0 & 0 \\ 0 & y-scale & 0 & 0 \\ 0 & 0 & z-scale & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.7}$$

where

$x-scale$ = rescaling along standard file x dimension

$y-scale$ = rescaling along standard file y dimension

$z-scale$ = rescaling along standard file z dimension

10.1.6 Perspective

Perspective distortion is achieved by applying the 4x4 homogenous coordinate transformation matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/xview & 1/yview & 1/zview & 1 \end{bmatrix} \tag{10.8}$$

where

$xview$ = x coordinate from which image is viewed

$yview$ = y coordinate from which image is viewed

$zview$ = z coordinate from which image is viewed

10.2 Singular Value Decomposition

Let X denote an m x n matrix of real-valued data and *rank* r, where without loss of generality $m \geq n$, and therefore $r \leq n$. In the case of microarray data, x_{ij} is the expression level of the i^{th} gene in the j^{th} assay. The elements of the i^{th} row of X form the n-dimensional vector $\mathbf{g}_i$, which we refer to as the *transcriptional response* of the i^{th} gene. Alternatively, the elements of the j^{th} column of X form the m-dimensional vector $\mathbf{a}_j$, which we refer to as the *expression profile* of the j^{th} assay.

The equation for singular value decomposition of X is the following:

$$X = U \Sigma V^T \tag{10.9}$$

where U is an m x n matrix, Σ is an n x n diagonal matrix, and V^T is also an n x n matrix. The columns of U are called the *left singular vectors*, $\{\mathbf{u}_k\}$, and form an orthonormal basis for the assay expression profiles, so that $\mathbf{u}_i \cdot \mathbf{u}_j = 1$ for $i = j$, and $\mathbf{u}_i \cdot \mathbf{u}_j = 0$ otherwise. The rows of V^T contain the elements of the *right singular vectors*, $\{\mathbf{v}_k\}$, and form an orthonormal basis for the gene transcriptional responses. The elements of Σ are only nonzero on the diagonal, and are called the *singular values*. Thus, $\Sigma = \mathrm{diag}(s_1,\ldots,s_n)$. Furthermore, $s_k > 0$ for $1 \leq k \leq r$, and $s_i = 0$ for $(r+1) \leq k \leq n$. By convention, the ordering of the singular vectors is determined by high-to-low sorting of singular values, with the highest singular value in the upper left index of the S matrix. Note that for a square, symmetric matrix X, singular value decomposition is equivalent to diagonalization, or solution of the eigenvalue problem.

One important result of the SVD of X is that:

$$X^{(l)} = \sum_{k=1}^{l} u_k s_k v_k^T \tag{10.10}$$

is the closest rank-*l* matrix to *X*. The term "closest" means that $X^{(l)}$ minimizes the sum of the squares of the difference of the elements of *X* and $X^{(l)}$, $\sum_{ij}|x_{ij} - x^{(l)}_{ij}|^2$.

One way to calculate the SVD is to first calculate V^T and *S* by diagonalizing X^TX:

$$X^T X = V \sum{}^2 V^T \tag{10.11}$$

and then to calculate *U* as follows:

$$U = XV \sum{}^{-1} \tag{10.12}$$

where the (*r*+1),...,*n* columns of *V* for which $s_k = 0$ are ignored in the matrix multiplication of (10.12). Choices for the remaining *n-r* singular vectors in *V* or *U* may be calculated using the Gram-Schmidt orthogonalization process or some other extension method. In practice there are several methods for calculating the SVD that are of higher accuracy and speed

Relation to principal component analysis. There is a direct relation between PCA and SVD in the case where principal components are calculated from the *covariance matrix*. If one conditions the data matrix *X* by *centering* each column, then $X^TX = \Sigma_i \mathbf{g}_i \mathbf{g}_i^T$ is proportional to the covariance matrix of the variables of $\mathbf{g}_i$ (*i.e.*, the covariance matrix of the assays). By (10.11), diagonalization of X^TX yields V^T, which also yields the principal components of $\{\mathbf{g}_i\}$. So, the right singular vectors $\{\mathbf{v}_k\}$ are the same as the principal components of $\{\mathbf{g}_i\}$. The eigenvalues of X^TX are equivalent to s_k^2, which are proportional to the variances of the principal components. The matrix $U\sum$ then contains the *principal component scores*, which are the coordinates of the genes in the space of principal components.

If instead each row of *X* is centered, $XX^T = \Sigma_j \mathbf{a}_j \mathbf{a}_j^T$ is proportional to the covariance matrix of the variables of $\mathbf{a}_j$ (*i.e.* the covariance matrix of the genes). In this case, the left singular vectors $\{\mathbf{u}_k\}$ are the same as the principal components of $\{\mathbf{a}_j\}$. The s_k^2 are again proportional to the variances of the principal components. The matrix $\sum V^T$ again contains the principal component scores, which are the coordinates of the assays in the space of principal components.

10.2.1 Uniqueness of the optimal solution to the absolute orientation problem

We show that the best rotation *R* to (6.9) is unique. Let

$$M = U \sum V^T = s_1 u_1 v_1^T + s_2 u_2 v_2^T + s_3 u_3 v_3^T \tag{10.13}$$

be an SVD of M, where *U* and *V* are orthogonal matrices and $\sum$ is diagonal. The solution for R is VU^t. U, $\sum$, and V are unique 1) making the same permutation *P* of the columns of *U*, elements of $\sum$, and columns of *V* , or 2) changing the sign of the corresponding columns of *U* and *V*, or 3) replacing columns of *U* and *V* corresponding to repeated singular values by any orthonormal basis of the span defined by the columns. This corresponds to rotating the columns by an orthogonal matrix.

For a square matrix M with an SVD $M = U\sum V^t$, all three changes do not affect VU^t. Let the new SVD under any of these changes be $U'\sum' V'^T$. For rotation, let

$$U' = UT, \quad V' = VT \tag{10.13}$$

then

$$V'U'^T = VTT^TU^T = VU^T \tag{10.13}$$

since $TT^t = I$. The same reasoning can be applied to permutation since permutation matrices are special cases of rotation matrices. Changing signs of corresponding columns of *U* and *V* will not change VU^T since $VU^T = v_1 u_1^T + v_2 u_2^T + v_3 u_3^T$.

10.2.2 Closedness of SVD

Suppose that $M_k \to M$, that (U_k, Σ_k, V_k) is an arbitrary SVD of M_k, and that $(U_k, \Sigma_k, V_k) \to (U, \Sigma, V)$. To show that SVD, viewed as a point-to-set mapping, is closed, we must show that (U, Σ, V) is a SVD of M.

From the closedness of *SO(3)*, *U* and *V* are orthonormal matrices. Likewise, the set of diagonal matrices in $\wp\ell(3)$ is a closed subgroup and, hence, Σ is a diagonal matrix. Therefore, (U, Σ, V) is an SVD of some matrix $M' = U \Sigma V^T$. However, by the continuity of transposition and matrix multiplication, if $(U_k, \Sigma_k, V_k) \to (U, \Sigma, V)$, then $U_k \Sigma_k, V_k^T \to U \Sigma V^T$ and, hence, $M_k \to M'$. Therefore, *M=M'* and, consequently, (U, Σ, V) is an SVD of M.

10.3 Matlab and C code

10.3.1 GLSDC function

```
function [sys,x0,str,ts] =
sfunGLS3(t,x,u,flag,X0,Markers,MV_SamplingTime,CCDsideH,CCDsideV,focal,Step,M
inMark)

% S-function for gaussian least square position and orientation estimation

persistent nUsedMarkers

switch flag,
  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,

[sys,x0,str,ts,nUsedMarkers]=mdlInitializeSizes(X0,Markers,MV_SamplingTime);

  %%%%%%%%%%
  % Update %
  %%%%%%%%%%
   case 2,

[sys,nUsedMarkers]=mdlUpdate(t,x,u,Markers,CCDsideH,CCDsideV,focal,Step,MinMa
rk);

  %%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%
  case 3,
    sys=[x;nUsedMarkers];

  %%%%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%%%
  case 9,
    sys=mdlTerminate(t,x,u);

  %%%%%%%%%%%%%%%%%%%%
  % Unexpected flags %
  %%%%%%%%%%%%%%%%%%%%
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
```

```
end

% end sfuntmpl

%
%=============================================================================
=
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
=
%
function
[sys,x0,str,ts,nUsedMarkers]=mdlInitializeSizes(X0,Markers,MV_SamplingTime)
%
% call simsizes for a sizes structure, fill it in and convert it to a
% sizes array.

[components,nmarkers]=size(Markers);

sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 6;
sizes.NumOutputs     = 6+1;
sizes.NumInputs      = 2*nmarkers;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

nUsedMarkers=-1;

x0=X0;
str = [];
ts  = [MV_SamplingTime 0];

% end mdlInitializeSizes

%
%=============================================================================
=
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
=
%
function
[x,nUsedMarkers]=mdlUpdate(t,x,u,Markers,CCDsideH,CCDsideV,focal,Step,MinMark
)

Xk=x;
for k=1:Step,
     [Xk,nUsedMarkers]=gales1(Xk,u,Markers',focal,CCDsideH,CCDsideV,MinMark);
end
x=Xk;

% end mdlUpdate
```

```
%
%=============================================================================
=
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
=
%
function sys=mdlTerminate(t,x,u)
sys = [];

% end mdlTerminate


function [Xknew,nUsedMarkers] =
gales1(Xk,Gk,Markers,f,CCDsideH,CCDsideV,MinMark)

N=size(Markers,1);
W=eye(2*N);

% calculate projections and gradients
A=derivatG(Markers,Xk,f);
Gs=computG(Markers,Xk,f);

% indexes to consider only markers inside the camera CCD
MkIn= (abs(Gk(1:2:2*N)) < ones(N,1)*CCDsideH) .* (abs(Gk(2:2:2*N)) <
ones(N,1)*CCDsideV);
UVIn=reshape(repmat(MkIn,1,2)',2*N,1);
ind=find([UVIn].*[1:2*N]'>0);

% consider the right column and rows in A and W
A=A(ind,:);
W=W(ind,ind);

% calculate difference in G
deltaGk=diag([UVIn])*(Gk-Gs);
deltaGk=deltaGk(ind);

% descent
P=A'*W*A;
dXk=pinv(P)*A'*W*deltaGk;

% number of used markers
nUsedMarkers=size(A,1)/2;

% update only if the number of markers is above the minimum
if ( nUsedMarkers > (MinMark-1) ),
    Xknew=Xk+dXk;
else
    Xknew=Xk;
end


%Calcolo della matrice A delle derivate di G
%rispetto alle coordinate e all'orientazione del centro del cestello
function AA=derivatG(Markers,Xk,f)

N=size(Markers,1);
AA=zeros(2*N,6);
A=zeros(2,6);
```

```
for ii=1:N,

    Point=Markers(ii,:);

    xM=Point(1);
    yM=Point(2);
    zM=Point(3);

    xD=Xk(1);
    yD=Xk(2);
    zD=Xk(3);
    psi=Xk(4);
    theta=Xk(5);
    phi=Xk(6);

    t1 = sin(psi);
    t2 = cos(theta);
    t3 = t1*t2;
    t4 = t3*xM;
    t5 = cos(psi);
    t6 = cos(phi);
    t7 = t5*t6;
    t8 = sin(theta);
    t9 = t1*t8;
    t10 = sin(phi);
    t11 = t9*t10;
    t14 = t5*t10;
    t15 = t9*t6;
    t16 = -t14+t15;
    t19 = t5*t2;
    t20 = t19*xM;
    t21 = t1*t6;
    t22 = t5*t8;
    t23 = t22*t10;
    t25 = (-t21+t23)*yM;
    t28 = t1*t10+t22*t6;
    t29 = t28*zM;
    t30 = xD+t20+t25+t29;
    t31 = t30*t30;
    t32 = 1/t31;
    t33 = (yD+t4+(t7+t11)*yM+t16*zM)*t32;
    t34 = 1/t30;
    t37 = -t7-t11;
    t41 = -t4+t37*yM+(t14-t15)*zM;
    t45 = t10*yM;
    t47 = t6*zM;
    t54 = -t22*xM+t19*t45+t19*t47;
    t64 = t28*yM+(t21-t23)*zM;
    t68 = t2*t10;
    t70 = t2*t6;
    t73 = (zD-t8*xM+t68*yM+t70*zM)*t32;
    A(1,1) = -t33;
    A(1,2) = t34;
    A(1,3) = 0.0;
    A(1,4) = (t20+t25+t29)*t34-t33*t41;
    A(1,5) = (-t9*xM+t3*t45+t3*t47)*t34-t33*t54;
    A(1,6) = (t16*yM+t37*zM)*t34-t33*t64;
    A(2,1) = -t73;
    A(2,2) = 0.0;
    A(2,3) = t34;
    A(2,4) = -t73*t41;
    A(2,5) = (-t2*xM-t8*t10*yM-t8*t6*zM)*t34-t73*t54;
    A(2,6) = (t70*yM-t68*zM)*t34-t73*t64;

    AA([2*ii-1,2*ii],1:6)=f*A;

end

% computation of the gradient of G
function G=computG(Markers,Xk,f)

N=size(Markers,1);
G=zeros(2*N,1);

for ii=1:N,

    Point=Markers(ii,:);
    xM=Point(1);
    yM=Point(2);
    zM=Point(3);
    xD=Xk(1);
    yD=Xk(2);
    zD=Xk(3);
    psi=Xk(4);
    theta=Xk(5);
    phi=Xk(6);

    t1 = sin(psi);
    t2 = cos(theta);
    t5 = cos(psi);
    t6 = cos(phi);
    t8 = sin(theta);
    t9 = t1*t8;
    t10 = sin(phi);
    t22 = t5*t8;
    t31 = 1/(xD+t5*t2*xM+(-t1*t6+t22*t10)*yM+(t1*t10+t22*t6)*zM);
    Ui = (yD+t1*t2*xM+(t5*t6+t9*t10)*yM+(-t5*t10+t9*t6)*zM)*t31;
    Vi = (zD-t8*xM+t2*t10*yM+t2*t6*zM)*t31;

    G([2*ii-1,2*ii])=f*[Ui; Vi];

  end
```

10.3.2 LHM function

```
   function [sys,x0,str,ts] =
sfunLHM1(t,x,u,flag,X0,Markers,CCDsideH,CCDsideV,focal,tol,epsilon,met,T)
   % S-function che usa l'algoritmo LHM per stimare la posizione e gli angoli
di eulero

   persistent nUsedMarkers
   %persistent Option

   switch flag,
     %%%%%%%%%%%%%%%%%%
     % Initialization %
     %%%%%%%%%%%%%%%%%%
     case 0,
       [sys,x0,str,ts,nUsedMarkers]=mdlInitializeSizes(X0,Markers,T);

     %%%%%%%%%%
     % Update %
     %%%%%%%%%%
      case 2,

[sys,nUsedMarkers]=mdlUpdate(t,x,u,Markers,CCDsideH,CCDsideV,focal,tol,epsilo
n,met);

     %%%%%%%%%%%
     % Outputs %
     %%%%%%%%%%%
     case 3,
       sys=mdlOutput(t,x,u,nUsedMarkers);

     %%%%%%%%%%%%%
     % Terminate %
     %%%%%%%%%%%%%
     case 9,
       sys=mdlTerminate(t,x,u);

     %%%%%%%%%%%%%%%%%%%%
     % Unexpected flags %
     %%%%%%%%%%%%%%%%%%%%
     otherwise
         error(['Unhandled flag = ',num2str(flag)])

   end

   % end sfuntmpl

   %
   %=============================================================================
====
   % mdlInitializeSizes
   % Return the sizes, initial conditions, and sample times for the S-
function.
   %=============================================================================
====
   %
   function [sys,x0,str,ts,nUsedMarkers]=mdlInitializeSizes(X0,Markers,T)
   %
   % call simsizes for a sizes structure, fill it in and convert it to a
   % sizes array.

   [components,nmarkers]=size(Markers);

   sizes = simsizes;

   sizes.NumContStates  = 0;
   sizes.NumDiscStates  = 12;
   sizes.NumOutputs     = 6+1;
   sizes.NumInputs      = 2*nmarkers;
   sizes.DirFeedthrough = 0;
   sizes.NumSampleTimes = 1;   % at least one sample time is needed

   sys = simsizes(sizes);
```

```
    nUsedMarkers=-1;
    %  Option = tol
    %  Option.epsilon= epsilon
    % Option.method = 'SVD'
    rot=[0 1 0; 0 0 -1; 1 0 0];
    x0=[X0(1:9);rot*X0(10:12)];
    str = [];
    ts  = [T 0];

    % end mdlInitializeSizes

    %
    %=============================================================================
====
    % mdlUpdate
    % Handle discrete state updates, sample time hits, and major time step
    % requirements.
    %=============================================================================
====
    %
    function
[x,nUsedMarkers]=mdlUpdate(t,x,u,Markers,CCDsideH,CCDsideV,focal,tol,epsilon,
met)
    % Secondo le nostre considerazioni l'algoritmo LHM utilizza un sistema di
    % riferimento diverso da quello descritto nell'articolo e soprattutto non
è destrorso, il sistema di
    % riferimento che secondo noi viene utilizzato è:
    %                  /|\
    %                   |  y'=v'
    %                   |
    %                   |
    %                   |_______________>
    %     z' entrante nel foglio       x'=u'
    %
    % Questo spiega i problemi che si hanno anche in uscita con gli angoli
    %
    % Il nostro sistema di riferimento è il classico sistema aereonautico di
assi corpo:
    %                    __________________>
    %  x entrante nel   |                 y=u
    %      foglio       |
    %                   |
    %                  \|/z=v
    % La trasformazione rigida tra questi sistemi di riferimento è
    %  [x'; y';z']= R * [x; y; z]   dove R= [0 1 0; 0 0 -1; 1 0 0]
    %  [u';v']= [1 0; 0 -1] *[u;v]
    %
    % Secondo noi l'alg LHM restituisce una matrice di rotazione tra sistemi
di
    % riferimenti non destrorsi e questo spiega perchè non ritorni la
    % trasformazione per gli angoli di eulero, per fare in modo che questi
ritornino
    % si devono invertire alcuni segni e scambiare tra loro le definizioni
    % degli angoli stessi, mentre la trasformazione funziona per il vettore di
    % traslazione t come facilmente intuibile.
    %
    Xk=x;
    a=size(Markers);
    nMarkers=a(2);
    %seleziono i markers interni allo spazio della telecamera
```

```
   MkIn= (abs(u(1:2:2*nMarkers)) < ones(nMarkers,1)*CCDsideH) .*
(abs(u(2:2:2*nMarkers)) < ones(nMarkers,1)*CCDsideV);

   markVisti=sum(MkIn);

   if(markVisti>4)
       %sistemo gli ingressi 2D per l'algoritmo LHM dando soltanto le
coordinate dei LEDs visibili e riportando in metri
       Qp1=reshape(u,2,nMarkers);
       Qp(1,:)=Qp1(1,find([MkIn].*[1:nMarkers]'>0));
       Qp(2,:)=-Qp1(2,find([MkIn].*[1:nMarkers]'>0));  %cambio il segno
       Qp=Qp./focal;
       %seleziono solo i Markers visibili
       for i=1:3
           P(i,:)=Markers(i,find([MkIn].*[1:nMarkers]'>0));
       end
       %cambio sistema di riferimento da assi corpo della telecamera al
       %sistema descritto sopra
       P=[0 1 0; 0 0 -1; 1 0 0]*P;
       Option.tol=tol;
       Option.epsilon=epsilon;
       Option.method=met;
       Option.initR=reshape(Xk(1:9),3,3);  %inizializzo initR con la matrice
trovata al passo precedente

       [R, t, it, obj_err, img_err] = objpose(P, Qp, Option);  %Funzione LHM
originale
   %    it
   %    obj_err
   %    img_err
       XK1=[reshape(R,9,1);t]; %nuovo vettore di stato
   else
       XK1=Xk;
   end

   x=XK1;
   nUsedMarkers=markVisti;

   % end mdlUpdate
   function sys=mdlOutput(t,x,u,nUsedMarkers)
   R=reshape(x(1:9),3,3);
   rot=[0 0 1; 1 0 0; 0 -1 0];
    %R=rot*R;

   ypr(2) = atan2(R(2,3),R(3,3));
   ypr(1) = -asin(-R(1,3));
   ypr(3) = atan2(R(1,2),R(1,1));

   t=rot*x(10:12);
   sys = [t;ypr';nUsedMarkers];

    %sys = [reshape(R,9,1);t;nUsedMarkers];

   %
   %=======================================================================
====
   % mdlTerminate
   % Perform any end of simulation tasks.
   %=======================================================================
====
   %
   function sys=mdlTerminate(t,x,u)
   sys=[];
   % end mdlTerminate



   function [R, t, it, obj_err, img_err] = objpose(P, Qp, options)
   % OBJPOSE - Object pose estimation
   %   OBJPOSE(P, Qp) compute the pose (exterior orientation)
   %   between the 3D point set P represented in object space
   %   and its projection Qp represented in normalized image
   %   plane. It implements the algorithm described in "Fast
   %   and Globally Convergent Pose Estimation from Video
   %   Images" by Chien-Ping Lu et. al. to appear in IEEE
   %   Transaction on Pattern Analysis and Machine intelligence
   %
   %   INPUTS:
   %     P - 3D point set arranged in a 3xn matrix
   %     Qp - 2D point set arranged in a 2xn matrix
   %     options - a structure specifies certain parameters in the algorithm.
   %
   %      Field name       Parameter                              Default
   %
   %      OPTIONS.initR    initial guess of rotation              none
   %      OPTIONS.tol      Convergence tolerance:                 1e-5
   %                       abs(new_value-old_value)/old_value<tol
   %      OPTIONS.epsilon  lower bound of the objective function  1e-8
   %      OPTIONS.method   'SVD' use SVD for solving rotation     'QTN'
   %                       'QTN' use quaternion for solving
   %                       rotation
   %   OUTPUTS:
   %     R - estimated rotation matrix
   %     t - estimated translation vector
   %     it - number of the iterations taken
   %     obj_err - object-space error associated with the estimate
   %     img_err - image-space error associated with the estimate
   %
   % TOL = 1E-5;
   % EPSILON = 1E-8;
   %  METHOD = 'SVD';

   if nargin >= 3
     if isfield(options, 'tol')
       TOL = options.tol;
     end

     if isfield(options, 'epsilon')
       EPSILON = options.epsilon;
     end

     if isfield(options, 'method')
       METHOD = options.method;
     end
   end

   n = size(P,2);

   % move the origin to the center of P
   pbar = sum(P,2)/n;
```

```
for i = 1:n
  P(:,i) = P(:,i)-pbar;
end

Q(1:3,n) = 0;
for i = 1 : n
  Q(:,i) = [Qp(:,i);1];
end

% compute projection matrices
F(1:3,1:3,1:n) = 0;
V(1:3) = 0;
for i = 1:n
  V = Q(:,i)/Q(3,i);
  F(:,:,i) = (V*V.')/(V.'*V);
end

% compute the matrix factor required to compute t
tFactor = inv(eye(3)-sum(F,3)/n)/n;

it = 0;
if isfield(options, 'initR')  % initial guess of rotation is given
  Ri = options.initR;
  Sum(1:3,1) = 0;
  for i = 1:n
    Sum = Sum + (F(:,:,i)-eye(3))*Ri*P(:,i);
  end
  ti = tFactor*Sum;

  % calculate error
  Qi = xform(P, Ri, ti);
  old_err = 0;
  vec(1:3,1) = 0;
  for i = 1 : n
    vec = (eye(3)-F(:,:,i))*Qi(:,i);
    old_err = old_err + vec'*vec;
    %     old_err = old_err + dot(vec,vec);
  end

else % no initial guess; use weak-perspective approximation
  % compute initial pose estimate
  [Ri, ti, Qi, old_err] = abskernel(P, Q, F, tFactor, METHOD);
  it = 1;
end

% compute next pose estimate
[Ri, ti, Qi, new_err] = abskernel(P, Qi, F, tFactor, METHOD);
it = it + 1;

while (abs((old_err-new_err)/old_err) > TOL) & (new_err > EPSILON)

  old_err = new_err;

  % compute the optimal estimate of R
  [Ri, ti, Qi, new_err] = abskernel(P, Qi, F, tFactor, METHOD);
  it = it + 1;

end
```

```
R = Ri;
t = ti;
obj_err = sqrt(new_err/n);

if (nargout >= 5) % calculate image-space error
  Qproj = xformproj(P, Ri, ti);
  img_err = 0;
  vec(1:3,1) = 0;
  for i = 1:n
    vec = Qproj(i)-Qp(i);
    img_err = img_err + vec'*vec;
%     img_err = img_err + dot(vec,vec);
  end
end
img_err = sqrt(img_err/n);

% correct possible reflection w.r.t the projection center
if t(3) < 0
  R = -R;
  t = -t;
end

% get back to original refernce frame
t = t - Ri*pbar;

% end of OBJPOSE

function [R, t, Qout, err2] = abskernel(P, Q, F, G, method)
% ABSKERNEL -  Absolute orientation kernel
%   ABSKERNEL is the function for solving the
%   intermediate absolute orientation problems
%   in the inner loop of the OI pose estimation
%   algorithm
%
%   INPUTS:
%     P - the reference point set arranged as a 3xn matrix
%     Q - the point set obtained by transforming P with
%         some pose estimate (typically the last estimate)
%     F - the array of projection matrices arranged as
%         a 3x3xn array
%     G - a matrix precomputed for calculating t
%     method - 'SVD'  -> use SVD solution for rotation
%              'QTN' -> use quaterion solution for rotation
%
%
%   OUTPUTS:
%     R - estimated rotation matrix
%     t - estimated translation vector
%     Qout - the point set obtained by transforming P with
%         newest pose estimate
%     err2 - sum of squared object-space error associated
%         with the estimate

n = size(P,2);

for i = 1:n
  Q(:,i) = F(:,:,i)*Q(:,i);
end

% compute P' and Q'
```

```
pbar = sum(P,2)/n;
qbar = sum(Q,2)/n;
for i = 1:n
  P(:,i) = P(:,i)-pbar;
  Q(:,i) = Q(:,i)-qbar;
end

if method == 'SVD' % use SVD solution
  % compute M matrix
  M(1:3,1:3) = 0;
  for i = 1:n
    M = M+P(:,i)*Q(:,i).';
  end

  % calculate SVD of M
  [U,S,V] = svd(M);

  % compute rotation matrix R
  R = V*(U.');
elseif method == 'QTN' % use quaternion solution
  % compute M matrix
  A(1:4,1:4) = 0;
  for i = 1:n
    A = A + qmatQ([1;Q(:,i)]).'*qmatW([1;P(:,i)]);
  end

  % Find the largest eigenvalue of A
  eigs_options.disp = 0;
  [V,D] = eigs(A, eye(size(A)), 1, 'LM', eigs_options);

  % compute rotation matrix R from the quaternion that
  % corresponds to the largest egienvalue of A
  R = quat2mat(V);
end

Sum(1:3,1) = 0;
for i = 1:n
  Sum = Sum + F(:,:,i)*R*P(:,i);
end
t = G*Sum;

Qout = xform(P, R, t);

% calculate error
err2 = 0;
vec(1:3,1) = 0;
for i = 1 : n
  vec = (eye(3)-F(:,:,i))*Qout(:,i);
  err2 = err2 + vec'*vec;
%   err2 = err2 + dot(vec,vec);
end

% end of ABSKERNEL


function Q = xform(P, R, t)
% XFORM - Transform
%   XFORM(P, R, t) transform the 3D point set P by rotation
%   R and translation t
```

```
%

n = size(P,2);

Q(1:3,n) = 0;

for i = 1:n
  Q(:,i) = R*P(:,i)+t;
end

%end function


function Qp = xformproj(P, R, t)
% XFORMPROJ - Transform and project
%   XFORMPROJ(P, R, t) transform the 3D point set P by
%   rotation R and translation t, and then project them
%   to the normalized image plane

%

n = size(P,2);

Q(1:3,n) = 0;
Qp(1:2,n) = 0;

for i = 1:n
  Q(:,i) = R*P(:,i)+t;
  Qp(:,i) = Q(1:2,i)/Q(3,i);
end

%end function


function Q = qmatQ(q)
% QMATQ - Compute the Q matrix (4x4) of quaternion q
%

w = q(1); x = q(2); y = q(3); z = q(4);
Q = [w, -x, -y, -z;
     x, w, -z, y;
     y, z, w, -x;
     z, -y, x, w];

%end function

function W = qmatW(q)
% QMATW - Compute the W matrix (4x4) of quaternion q
%

w = q(1); x = q(2); y = q(3); z = q(4);
W = [w, -x, -y, -z;
     x, w, z, -y;
     y, -z, w, x;
     z, y, -x, w];

%end function
```

```
function R = quat2mat(q)
% QUAT2MAT - Convert a quaternion to a 3x3 rotation matrix
%

a = q(1); b = q(2); c = q(3); d = q(4);
R = [a^2+b^2-c^2-d^2, 2*(b*c-a*d), 2*(b*d+a*c); ...
     2*(b*c+a*d), a^2+c^2-b^2-d^2, 2*(c*d-a*b); ...
     2*(b*d-a*c), 2*(c*d+a*b), a^2+d^2-b^2-c^2];

%end function
```

10.3.3 Labeling function

```
/*  S-Function for Marker Labeling ** G.Campa & M.Mammarella ** November
2004 *****************/

#define S_FUNCTION_NAME labeling
#define S_FUNCTION_LEVEL 2


#include "simstruc.h"

/* mdlCheckParameters, check parameters, this routine is called later from
mdlInitializeSizes */
#define MDL_CHECK_PARAMETERS
static void mdlCheckParameters(SimStruct *S)
{
    /* Basic check : All parameters must be real positive vectors
*/
    real_T *pr;

    int_T  i, el, nEls;
    for (i = 0; i < 5; i++) {
        if (mxIsEmpty(    ssGetSFcnParam(S,i)) || mxIsSparse(
ssGetSFcnParam(S,i)) ||
            mxIsComplex(  ssGetSFcnParam(S,i)) || !mxIsNumeric(
ssGetSFcnParam(S,i))   )
                  { ssSetErrorStatus(S,"Parameters must be real finite
vectors"); return; }
        pr   = mxGetPr(ssGetSFcnParam(S,i));
        nEls = mxGetNumberOfElements(ssGetSFcnParam(S,i));
        for (el = 0; el < nEls; el++) {
            if (!mxIsFinite(pr[el]))
                  { ssSetErrorStatus(S,"Parameters must be real finite
vectors"); return; }
        }
    }

    /* Check number of elements in parameter: nmarker
*/
    if ( mxGetNumberOfElements(ssGetSFcnParam(S,0)) != 1 )
    { ssSetErrorStatus(S,"The parameter must be a scalar"); return; }

    /* get the basic parameters and check them
*/
    pr=mxGetPr(ssGetSFcnParam(S,0));
    if ( pr[0] < 1 )
```

```
        { ssSetErrorStatus(S,"Number of markers must be greater than zero");
return; }


        /* Check number of elements in parameter: max number of point
*/
        if ( mxGetNumberOfElements(ssGetSFcnParam(S,1)) != 1 )
        { ssSetErrorStatus(S,"The parameter must be a scalar"); return; }

        /* get the basic parameters and check them
*/
        pr=mxGetPr(ssGetSFcnParam(S,1));
        if ( pr[0] < 1 )
        { ssSetErrorStatus(S,"Max number of point must be greater than zero");
return; }

            /* Check number of elements in parameter: focal length
*/
        if ( mxGetNumberOfElements(ssGetSFcnParam(S,2)) != 1 )
        { ssSetErrorStatus(S,"The parameter must be a scalar"); return; }

        /* get the basic parameters and check them
*/
        pr=mxGetPr(ssGetSFcnParam(S,2));
        if ( pr[0] < 0 )
        { ssSetErrorStatus(S,"Focal length cannot be negative"); return; }

        /* Check number of elements in parameter: screen limit
*/
        if ( mxGetNumberOfElements(ssGetSFcnParam(S,3)) != 4 )      // screen
limit
        { ssSetErrorStatus(S,"The screen limit must be a 4 elements vector");
return; }


         /* Check number of elements in parameter: sampling time
*/
        if ( mxGetNumberOfElements(ssGetSFcnParam(S,4)) != 1 )
        { ssSetErrorStatus(S,"The parameter must be a scalar"); return; }

        /* get the basic parameters and check them
*/
        pr=mxGetPr(ssGetSFcnParam(S,4));
        if ( pr[0] < 0 )
        { ssSetErrorStatus(S,"Sampling Time cannot be negative"); return; }




    }

    /* mdlInitializeSizes - initialize the sizes array
********************************************/
    static void mdlInitializeSizes(SimStruct *S)
    {
        real_T *n, *m;

        n=mxGetPr(ssGetSFcnParam(S,0));                // number of markers
        m=mxGetPr(ssGetSFcnParam(S,1));                // number of max point
from input
```

```
        ssSetNumSFcnParams(S,5);                           /* number of
expected parameters         */

        /* Check the number of parameters and then calls mdlCheckParameters to
see if they are ok */
        if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S))
        { mdlCheckParameters(S); if (ssGetErrorStatus(S) != NULL) return; }
else return;
        n=mxGetPr(ssGetSFcnParam(S,0));

        ssSetNumContStates(S,0);                           /* number of
continuous states           */
        ssSetNumDiscStates(S,0);                           /* number of
discrete states             */

        if (!ssSetNumInputPorts(S,2)) return;              /* number of input
ports                  */
        ssSetInputPortWidth(S,0,(int_T)(n[0]*4));          /* first input port
width                  */
        ssSetInputPortWidth(S,1,(int_T)(m[0]*2));          /* second input port
width                 */
        ssSetInputPortDirectFeedThrough(S,0,1);            /* first port direct
feedthrough flag    */
        ssSetInputPortDirectFeedThrough(S,1,1);            /* second port
direct feedthrough flag   */

        if (!ssSetNumOutputPorts(S,2)) return;             /* number of output
ports                  */
        ssSetOutputPortWidth(S,0,(int_T)(n[0]*2));         /* first output port
width                 */
        ssSetOutputPortWidth(S,1,(int_T)(2*n[0]));         /* second output
port width                */

        ssSetNumSampleTimes(S,0);                          /* number of sample
times                  */

        ssSetNumRWork(S,(int_T)(n[0]*m[0]+n[0]+m[0]));       /* number real
work vector elements      */
        ssSetNumIWork(S,m[0]);                          /* number int_T work
vector elements     */
        ssSetNumPWork(S,0);                                /* number ptr work
vector elements       */
        ssSetNumModes(S,0);                                /* number mode work
vector elements      */
        ssSetNumNonsampledZCs(S,0);                        /* number of
nonsampled zero crossing    */
    }

    /* mdlInitializeSampleTimes - initialize the sample times array
*******************************/
    static void mdlInitializeSampleTimes(SimStruct *S)
    {
        real_T *t;
        t=mxGetPr(ssGetSFcnParam(S,4));

        ssSetSampleTime(S, 0, *t);
        ssSetOffsetTime(S, 0, 0);
    }
```

```
   /* mdlStart - initialize work vectors
**********************************************************/
   #define MDL_START
   static void mdlStart(SimStruct *S)
   {
       int_T i,j;
       real_T *nMark,*nPoint;
       real_T *errM;
       nMark=mxGetPr(ssGetSFcnParam(S,0));
       nPoint=mxGetPr(ssGetSFcnParam(S,1));       // max num di punti
       errM= ssGetRWork(S);

       for (i=0;i<nMark[0]+1;i++)
           for(j=0;j<nPoint[0]+1;j++)             // inizializzo anche i
vettori di minimo
               errM[j*(int_T)(nMark[0])+i]=1000;
   }

   /* mdlOutputs - compute the outputs
************************************************************/
   static void mdlOutputs(SimStruct *S, int_T tid)
   {
   int_T     i, j, k;
   real_T             *y1  = ssGetOutputPortRealSignal(S,0);
   real_T             *y2  = ssGetOutputPortRealSignal(S,1);
   InputRealPtrsType up1  = ssGetInputPortRealSignalPtrs(S,0);
   InputRealPtrsType up2  = ssGetInputPortRealSignalPtrs(S,1);
   real_T *focal;
   real_T *nMarkR;
   int_T nMark;
   real_T *nPointR;
   int_T nPoint;
   real_T *errM;
   real_T *minR;
   real_T *minC;
   int_T *indiciC;
   real_T *limit;

   // y[2] y è puntatore a vettore;
   // (*up[2]) up è puntatore a puntatore


   nMarkR=mxGetPr(ssGetSFcnParam(S,0));
   nPointR=mxGetPr(ssGetSFcnParam(S,1));
   focal=mxGetPr(ssGetSFcnParam(S,2));
   limit=mxGetPr(ssGetSFcnParam(S,3));

   nMark=(int_T)(nMarkR[0]);
   nPoint=(int_T)(nPointR[0]);


   // si implementa la camera simulata

   for (i=0;i<nMark;i++)
       {
   //  Xi[i]= (*up1[4*i]); (asse di profondità della camera)
   //  Yi[i]= (*up1[4*i +1]); (asse che punta a destra)
   //  Zi[i]= (*up1[4*i +2 ]); (asse che punta verso il basso)
```

```
        if ((*up1[4*i]) !=0 )       // division by zero checking
            {
            y1[2*i]=   focal[0] * (*up1[4*i+ 1])/(*up1[4*i]);
            y1[2*i+1]= focal[0] * (*up1[4*i+ 2])/(*up1[4*i]);
            }
        else
            y1[2*i]=y1[2*i+1]=0;
        }


    // labeling dei markers

    errM= ssGetRWork(S);
    minR = &errM[nMark*nPoint]; //il vettore di minimo di riga inizia alla
posizione nMark*nPoint
                                    //ed ha nMark posizioni
    minC= &minR[nMark];         //il vettore di minimo di colonna inizia alla
posizione nMark
                                    //ed ha nPoint posizioni

    indiciC=ssGetIWork(S);          // vettore indici delle colonne ha nPoint
posizioni

    for (i=0;i<nMark;i++)
        {
        minR[i]=1000;
        for(j=0;j<nPoint;j++)             // inizializzo errM e minR e minC a
1000
                {
                errM[j*nMark+i]=1000;           // questo va fatto per non
confondersi con il passo precedente
                minC[j]=1000;
                }
        }
    for (i=0;i< nMark;i++)
        {
        for (j=0; j<nPoint;j++)
            {
                if ((  *up2[2*j]  >= limit[0]) && ( *up2[2*j]  <= limit[1] )
&& (  *up2[2*j+1]  >= limit[2]) && ( *up2[2*j+1]  <= limit[3] ) )
                    {                                                   //
calcolo le distanze se i markers trovati rientrano nello schermo
                    errM[j*nMark +i]=  ( (*up2[2*j] - y1[2*i])*(*up2[2*j] -
y1[2*i])+
                                            (*up2[2*j+1] - y1[2*i
+1])*(*up2[2*j+ 1] - y1[2*i +1]));
                    }
            }
        }

    for (i=0; i< nMark; i++)
        {
        for (j=0;j<nPoint; j++)
            {
            if (errM[j*nMark +i] < minR[i])
                minR[i]=errM[j*nMark +i];

            if (errM[j*nMark +i] < minC[j])
                {
                minC[j]=errM[j*nMark +i];
                indiciC[j]=i+1;
                }
            if (minC[j]==1000)
                indiciC[j]=0;
            }

        }

    for (i=0;i<nMark;i++)
        {
            y2[2*i]=100;                    // inizializzo output altrimenti
rimane il valore del passo
            y2[2*i+1]=100;                  // precedente
        }



    for (j=0;j<nPoint; j++)
        {
        if(indiciC[j]>0)
            {
            if(minC[j] == minR[ (indiciC[j]-1) ])
                {
                y2[2*(indiciC[j]-1)]= *up2[2*j];
                y2[(2*(indiciC[j]-1)) +1]= *up2[2*j +1];
                }
            }
    }

    }

    /* mdlTerminate - called when the simulation is terminated
***********************************/
    static void mdlTerminate(SimStruct *S) {}

    /* Trailer information to set everything up for simulink usage
*******************************/
    #ifdef  MATLAB_MEX_FILE                       /* Is this file being
compiled as a MEX-file?   */
    #include "simulink.c"                         /* MEX-file interface
mechanism                 */
    #else
    #include "cg_sfun.h"                          /* Code generation
registration function        */
    #endif
```