

Multiple Target Tracking System Design for Driver Assistance Application

Jehangir Khan Smail Niar Atika Menhaj Yassin Elhillali

Université de Valenciennes et du Hainaut Cambrésis, France

[/jehangir.khan, smail.niar, atika.menhaj, yassin.elhillali}@univ-valenciennes.fr](mailto:{jehangir.khan, smail.niar, atika.menhaj, yassin.elhillali}@univ-valenciennes.fr)

Abstract --- This paper discusses the design of an entire Multiple Target Tracking (MTT) system. Use of MTT in driver assistant systems makes them very efficient and effective in collision avoidance and early warning. We describe the procedure that we chose for implementing an MTT system on a reconfigurable platform. We also examine in detail each of the task composing the MTT chain. In our implementation, several independent parallel tasks have been identified and mapped onto a multiprocessor architecture to achieve the deadlines imposed by the application. An Automotive-radar is used as the front end sensor in our application. We also take into account the constraints imposed by our embedded system. This study demonstrates that the joint utility of reconfigurable circuits (namely FPGA) and MPSoC, facilitates the development of a flexible and efficient MTT system.

I. INTRODUCTION

Unfavorable weather, low visibility conditions, misjudgment of delicate situations and physical or mental stress are among the reasons that put the lives of the driver and his/her passengers in danger. Relieving the driver of the stressful driving conditions guarantees a drop in road accidents. Driver Assistant Systems (DAS) help drivers take correct and quick decisions in delicate situations. These systems provide the driver a realistic assessment of the dynamic behavior of potential obstacles well before it is too late to react.

Use of Multiple-Target Tracking (MTT) enhances the affectivity of driver assistance systems to aid drivers in taking correct decisions in critical situations. The purpose of target tracking is to collect data from the sensor field of view (FOV) containing one or more potential targets of interest and to partition the sensor data into sets of observations, or tracks [1]. In context of driver assistance, a target tracking system detects and monitors the dynamic behavior of one or more obstacles in the way of the host vehicle.

Our implementation of the MTT system has two main features among others. First, being radar-based, it has the advantages of longer range as compared to camera based systems. It performs better in bad visibility conditions and has lower computational requirements [3]. Radar based multi-target tracking is achievable with relatively more ease and less complexity as compared to camera based tracking.

Moreover, radar helps detect obstacles at longer distances and hence ensures longer reaction time for vehicle drivers.

Secondly, we implement our system on FPGA using MPSoC architecture which is inherently flexible and adaptable. This feature capitalizes on advances in FPGA fabrication technology allowing a cost effective implementation of complex embedded applications. Charting of processor properties over the last three decades shows that the performance of a single processor has leveled off in the last decade[13]. Dedicated hardware implementation may be useful for high speed processing but it does not offer the flexibility and programmability desired for system evolution. Applications with tight resource-consumption and runtime constraints are increasingly resorting to MPSoC architectures. The move to MPSoC design elegantly addresses the power issues faced on the hardware side. Creating multiple processors that execute at lower frequency, results in comparable overall performance in terms of instructions per second while allowing designers to slow down the clock speed, which is a major constraint for low power designs [13].

Many studies have been done on the isolated parts of MTT system [2, 3 7,8] but implementation of the complete MTT system on a reconfigurable platform and its application to automotive safety is rather rare.

II. THE APPLICATION

A. Terminology

In context of target tracking applications, a *target* represents an obstacle in the way of the host vehicle. With every obstacle is associated a *state* which is represented as a vector that contains parameters defining target's position and its dynamics in space e.g. its distance, speed, azimuth or elevation etc. A state vector with n elements is called n -state vector. A concatenation of target states defining the target trajectory or movement history at discrete moments in time is called a *track*. As detailed below in MTT, tracking deals with 3 quantities: the *Observation*, which corresponds to the measurement of a target's state by a sensor (radar) at discrete moments in time. It is one of the two representations of the true state of the target. The other representation is a calculated "guess" or *prediction* of the

target's true state before the observation arrives. Taking into account the *observation* and the *prediction*, an *estimate* about the true target state is made. Estimate is the corrected state of the target that depends upon the variances of both the observation and the prediction. In this paper, the term *scan* is used to name the periodic sweep of radar FOV giving observations of all the detected targets.

B. MTT building blocks

A simplified view of Multiple Target Tracking (MTT) system is given in figure 1. The system can broadly be divided into two main functions namely *Data Association* and *Filtering & Prediction*. The two functions work in a close loop. The data association function is further divided into three sub-functions; "*Track maintenance*", "*Observation-to-Track Assignment*" and "*Gate Computation*". Detailed description of these functions and sub-functions is given in the next section.

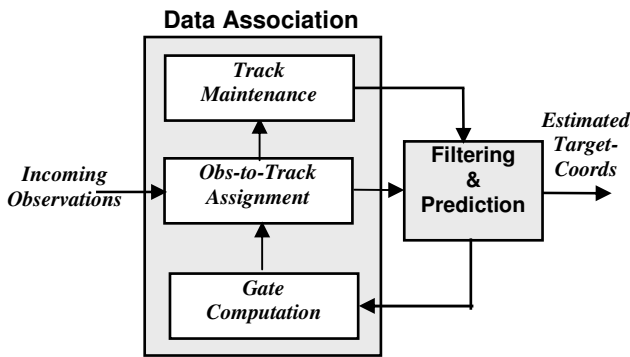


Figure 1: A simplified view of MTT

III. APPLICATION DEVELOPMENT

For the purpose of parallelized implementation we organized the application into sub-modules as shown in figure 2. The functioning of the system is explained as follows.

Assuming recursive processing as shown by the loop in figure 1, tracks would have been formed on the previous radar *scan*. When new observations are received from the sensor the processing loop is to be executed. Incoming observations are first considered by the "*Gate checker*" for updating of the existing tracks. Gating tests determine which possible "*observation-to-track*" pairings are reasonable, by attributing a cost to each pairing. The costs are calculated as the statistical distance between the *predictions* of the target states given by the filters and the *observed state* coordinates received from the radar. These

costs are put together in a *cost matrix* which is then passed on to the *assignment solver* to determine the finalized pairings. The pairings are made in a way to ensure minimum total cost for all the pairings. The finalized observation-track pairings are passed on to the tracking filters which use them for estimating the current states of targets and predicting the next states as well as the *error covariance* associated with these predictions.

The predicted states and predicted error covariance are used by the "*Gate compute*" function to define probability *gates* or windows around the predicted states. The dimensions of the gates being dictated by the prediction error covariance, these gates demarcate the probability boundaries for the next state coordinate measurements. The "*Gate Compute*" sub-function can be viewed as a first level of "*screening out*" the unlikely target-track associations in case of multiple observations falling close to a single prediction or vice versa. In the second level of "*screening*", namely observation-to-track assignment, a strictly one-to-one coupling is established between observations and tracks.

The "*Track Maintenance*" sub-function consists of three blocks. The "*obs-less Gate Identifier*" identifies the gate where no observation falls. This indicates a probable disappearance of an already known target and hence the deletion of its track after confirmation. The "*New Target Identifier*" detects observations that fall outside all the gates. These observations are potential candidates for initiating new tracks after confirmation. The "*Track Init/Del*" block initiates new tracks or deletes existing ones when needed. In context of this work, 3 observations out of 5 scans for the same target initiate a new track while 3 consecutive misses out of 5 scans for an existing target prompts the deletion of its track. The "*Tracking filters*" block in figure 2, is particularly important. We use Kalman filters for this block. The number of filters employed is equal to the maximum number of targets to be tracked. In our current work we have fixed this number at 10. In the final system we will increase it up to 20 as the radar we are using can measure the coordinates of a maximum of 20 targets. Hence this block will use 20 similar filters in final system.

At start up, at most 10 of the "*incoming observations*" would simply pass through the "*Gate Checker*", "*Cost Matrix Generator*" and "*Assignment Solver*" on to the filters' inputs. The filter takes an observation as an "*inaccurate*" representation of the "*true state*" of the target and the amount of inaccuracy of the observation depends on the measurement variance of the sensor. The filter then estimates the current state of the target and predicts its next state before the next observation is available. To estimate the true state we need a *process model*, a *measurement model* and an *estimator*. These are all detailed below.

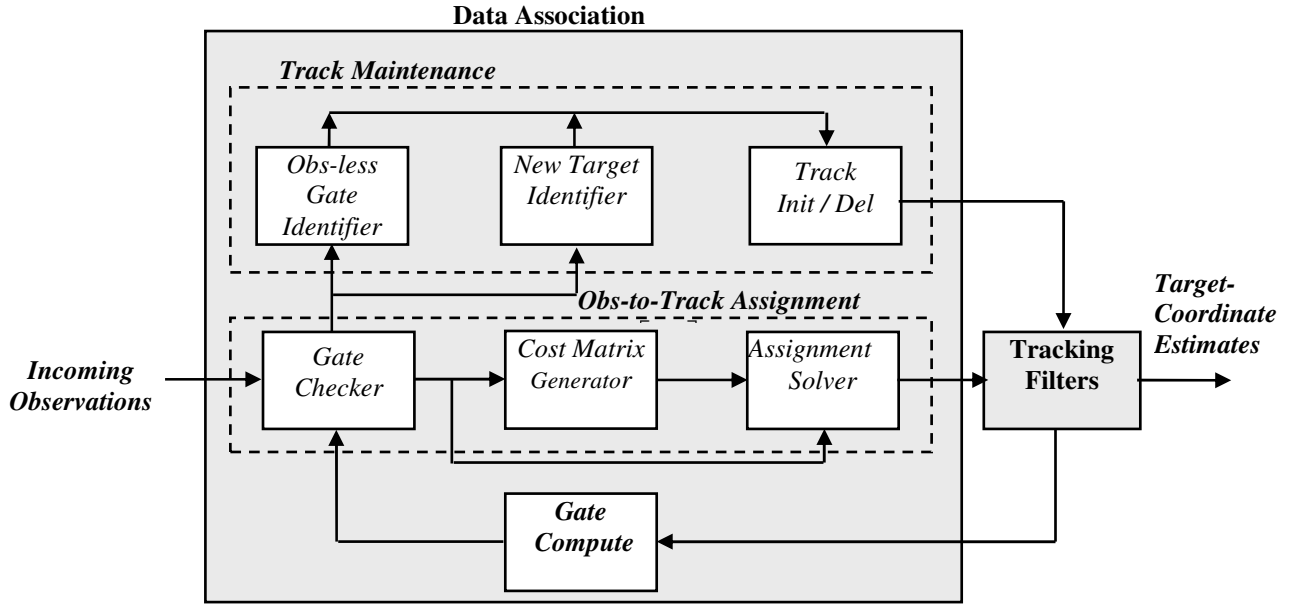


Figure 2: The Proposed MTT implementation

A. Process Model

The *process model* mathematically projects the process current state to the future. This can be presented in a linear stochastic difference equation as

$$Y_k = A Y_{k-1} + B U_k + W_{k-1} \quad (3.1)$$

In equation (3.1) Y_{k-1} and Y_k are n -dimensional state vectors that includes the quantities to be estimated. Vector Y_{k-1} represents the state at instant $k-1$ while Y_k represents the state at instant k . The $n \times n$ matrix A in the difference equation (3.1) relates the state at time step $k-1$ to the state at time step k , in the absence of either a driving function or process noise. Matrix A is the assumed known *state transition matrix* which may be viewed as the coefficient of state transformation from instant $k-1$ to instant k , in absence of any driving signal and process noise. The $n \times l$ matrix B relates the optional control input $U_k \in \mathcal{R}^l$ to the state Y_k whereas W_{k-1} is zero-mean additive white Gaussian process noise (AWGN) with assumed known covariance Q . Matrix B is the assumed known *control matrix* and U_k is the deterministic input, such as the relative position change associated with the host-vehicle (own ship) motion.

B. Measurement Model

To describe the relationship between the true state and the measurements (observations) a *measurement model* is required. It can be described as a linear expression

$$Z_k = H Y_k + V_k \quad (3.2)$$

Here Z_k is the measurement or observation vector containing two elements distance d and angle θ as shown below. The $m \times n$ *observation matrix* H in the measurement

equation (3.2) relates the current state to the *measurement (observation) vector* Z_k . The terms V_k in equation (3.2) is a random variable representing the measurement noise.

For implementation we chose the example case given in [2]. In this example the matrices and vectors in equations (3.1) and (3.2) have the forms shown below. In the rest of the paper the numerical values of all the matrix and vector elements are borrowed from this example. Quantity T is equal to 0.02 seconds and it is the radar Pulse Repetition Time (PRT) specific to the radar unit we are using in our project. Y_k , A and Z_k have the following forms:

$$Y_k = \begin{bmatrix} y_{11} \\ y_{21} \\ y_{31} \\ y_{41} \end{bmatrix} \quad A = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Z_k = \begin{bmatrix} d \\ \theta \end{bmatrix}$$

Here y_{11} is the target range or distance, y_{21} is range rate or speed, y_{31} is angle (azimuth), y_{41} is angle rate or angular speed. In vector Z_k the element d is the distance measurement and θ is the azimuth angle measurement. Matrix B and control input U_k are ignored here because they are not necessary in our application.

Having devised the process and measurement models, we need an *estimator* which would use these models to estimate the true state. Our solution is based on the Kalman filter which is a recursive Least Square Estimator (LSE) and is considered to be the optimal estimator for linear systems [4,5].

C. Kalman Filter

Many good derivations of the Kalman filter and discussions of its applications are presented in the literature [4,5,6], so only the resulting equations are given here.

Given the process and the measurement models from (3.1)

and (3.2), the Kalman filter equations are

$$\hat{Y}_k^- = A \hat{Y}_{k-1} + B U_k \quad (3.3a)$$

$$P_k^- = A P_{k-1} A^T + Q \quad (3.3b)$$

$$K = P_k^- H^T (H P_k^- H^T + R)^{-1} \quad (3.3c)$$

$$\hat{Y}_k = \hat{Y}_k^- + K (Z_k - H \hat{Y}_k^-) \quad (3.3d)$$

$$P_k = (I - KH) P_k^- \quad (3.3e)$$

Here \hat{Y}_k is state estimation vector, \hat{Y}_k^- is state prediction vector, K is Kalman gain matrix, P_k^- is prediction error covariance matrix, P_k is estimation covariance matrix and I is an identity matrix of the same dimensions as P_k .

The newly introduced vectors and matrices in equations (3.3) have the following forms.

$$\hat{Y}_k = \begin{bmatrix} \hat{y}_{11} \\ \hat{y}_{21} \\ \hat{y}_{31} \\ \hat{y}_{41} \end{bmatrix} \quad \hat{Y}_k^- = \begin{bmatrix} \hat{y}_{11}^- \\ \hat{y}_{21}^- \\ \hat{y}_{31}^- \\ \hat{y}_{41}^- \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad R = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} 10^6 & 0 \\ 0 & 2.9 * 10^{-4} \end{bmatrix}$$

$$Q = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ q_{41} & q_{42} & q_{43} & q_{44} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 330 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.3 * 10^{-8} \end{bmatrix}$$

Here \hat{y}_{11}^- is range Prediction \hat{y}_{21}^- is speed prediction \hat{y}_{31}^- is azimuth angle prediction, \hat{y}_{41}^- is angular speed prediction \hat{y}_{11} is range estimate, \hat{y}_{21} speed estimate, \hat{y}_{31} is angle estimate and \hat{y}_{41} is angular speed estimate, all for instant k .

Matrices K and P_k^- have the following forms.

$$K = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \\ k_{31} & k_{32} \\ k_{41} & k_{42} \end{bmatrix} \quad P_k^- = \begin{bmatrix} P_{11}^- & P_{12}^- & P_{13}^- & P_{14}^- \\ P_{21}^- & P_{22}^- & P_{23}^- & P_{24}^- \\ P_{31}^- & P_{32}^- & P_{33}^- & P_{34}^- \\ P_{41}^- & P_{42}^- & P_{43}^- & P_{44}^- \end{bmatrix}$$

Matrix P_k is similar in form to P_k^- except for the superscript '-'. The scan index k has been ignored in the elements of these matrices and vectors for the sake of notational simplicity.

The Kalman filter cycles through the "prediction-correction" loop shown pictorially in figure 3. In the prediction step (also called *time update*), the filter predicts the next state and error covariance associated with the state prediction using equations (3.3a) and (3.3b) respectively. In

the correction step (also called *measurement update*), the filter calculates the filter gain and estimates the current state and the error covariance of this estimation using equations (3.3c) through (3.3e).

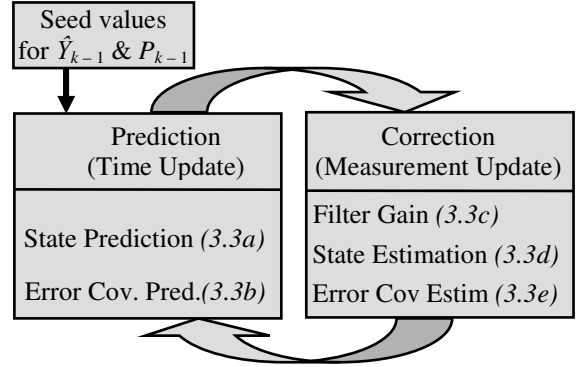


Figure 3: The Kalman filter

Figure 4 shows the state (position) of a target estimated by the Kalman filter against the true position and that measured by the radar. Notice how closely the estimated position follows the real position as compared with the measured position after the 20 transitional iterations.

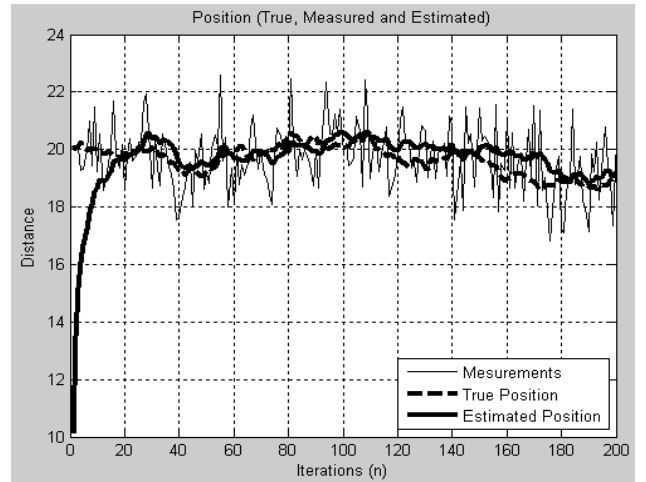


Figure 4: Estimated target position

As we are dealing with multiple targets at the same time, we have to identify which of the incoming observed states to associate with which of the predicted states for making the estimations. This is the job of *data association* function. The data association sub-modules are explained in the following paragraphs.

D. Gate Computation

The first step in data association is gate computation. The "Gate Compute" block receives \hat{Y}_k^- and P_k^- from the Kalman Filters for all the tracks. Using these two quantities the "Gate Compute" block defines the probability gates or windows which are used to verify whether an incoming

observation can be associated with an existing track. The gate computation process is summarized below.

Define \tilde{Y} to be the *residual* or *innovation* vector which is the difference between the actual measurement Z_k and the expected (predicted) measurement vector $[H\hat{Y}_k^-]$. In general at scan k ,

$$\tilde{Y}_i = Z_i - H\hat{Y}_i^- \quad (3.4)$$

Time index k is ignored for simplicity. Now define a rectangular region such that an observation vector Z_k (with elements $z_{k\perp}$) is said to satisfy the gate of a given track if all elements \tilde{Y}_{il} of residual vector \tilde{Y}_i satisfy the relationship

$$|Z_{k\perp} - H\hat{Y}_{il}^-| = |\tilde{Y}_{il}| \leq K_{Gl}\sigma_r \quad (3.5)$$

Here i is an index for track i , G signifies for gate and l is replaced either by d or by θ , whichever is appropriate (see equations 3.10 and 3.11). The term σ_r is the *residual standard deviation* and is defined in terms of the *measurement variance* σ_z^2 and *prediction variance* $\sigma_{\hat{y}_k}^2$. Typical choice for K_{Gl} is $[K_{Gl} \geq 3.0]$. This large choice of gating coefficient is typically made in order to compensate for the approximations involved in modeling the target dynamics through the Kalman filter covariance matrix [1]. This concept comes from the famous “3 sigma rule” in statistics.

In its matrix form equation (3.4) can be simplified down to

$$\tilde{Y}_i = \begin{bmatrix} \tilde{y}_{i11} \\ \tilde{y}_{i21} \end{bmatrix} = \begin{bmatrix} d_i - \hat{y}_{11}^- \\ \theta_i - \hat{y}_{31}^- \end{bmatrix} \quad (3.6)$$

Consequently equation (3.5) gives

$$\begin{bmatrix} \tilde{y}_{i11} \\ \tilde{y}_{i21} \end{bmatrix} \leq K_{Gl}\sigma_r \quad (3.7)$$

The residual standard deviations for the two state vector elements are defined as follows

$$\sigma_{rd} = \sqrt{r_{11} + p_{22}^-} \quad (3.8)$$

$$\sigma_{r\theta} = \sqrt{r_{22} + p_{44}^-} \quad (3.9)$$

From (3.7), (3.8) and (3.9) we get

$$|\tilde{y}_{i11}| = |\tilde{y}_{id}| \leq 3.0\sqrt{r_{11} + p_{22}^-} \quad (3.10)$$

$$|\tilde{y}_{i21}| = |\tilde{y}_{i\theta}| \leq 3.0\sqrt{r_{22} + p_{44}^-} \quad (3.11)$$

Equations (3.10) and (3.11) together put the limits on the residuals \tilde{y}_{id} and $\tilde{y}_{i\theta}$. In other words, the difference between an incoming observation and prediction for track i must obey equations (3.10) and (3.11) for the observation to be assigned to track i .

E. Gate Checker

The “Gate checker” tests whether an incoming observation fulfills the conditions set in equations (3.10) and (3.11). In effect, this block sets or resets the binary elements of an $N \times N$ matrix termed as the “Gate Mask” matrix M .

Here N is the maximum number of targets and tracks being processed. If an observation j fulfills both these conditions for a track i , the corresponding element m_{ij} of matrix M is set to 1 otherwise it is reset to 0.

Matrix M would typically have more than one 1’s in a column or a row. The ultimate goal is to have only one ‘1’ in a row or a column for a one-to-one coupling of observations and predictions.

$$M = \begin{matrix} & \underbrace{\hspace{10em}}_{\text{Tracks}} & \\ \left. \begin{matrix} m_{11} & m_{12} & \bullet & \bullet & \bullet & m_{1N} \\ m_{21} & m_{22} & \bullet & \bullet & \bullet & m_{2N} \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ m_{N1} & m_{N2} & \bullet & \bullet & \bullet & m_{NN} \end{matrix} \right\} & \text{Obs} \end{matrix}$$

$$m_{ij} = \begin{cases} 1 & \text{if obs } j \text{ obeys (3.10) \& (3.11) for track } i \\ 0 & \text{otherwise} \end{cases}$$

To achieve this goal, the first step is to attach a cost to every possible coupling. This is done by the “Cost Generator” block explained next.

F. Cost Matrix Generator

The “Cost Matrix Generator” associates a cost with every possible observation-prediction pairing. The cost c_{ij} for associating an observation j with a prediction i is the statistical distance d_{ij}^2 between the observation and the prediction when m_{ij} is 1. The cost is an arbitrarily large number when m_{ij} is 0.

The distance d_{ij}^2 is calculated as follows.

$$\text{Define } S_{ij} = H P_k^- H^T + R \quad (3.12)$$

Here j is an index for observation j and i is the index for track i in a scan, S_{ij} is the residual covariance matrix. The statistical distance d_{ij}^2 is the norm of the residual vector \tilde{Y}_{ij}

$$d_{ij}^2 = \tilde{Y}_{ij}^T S_{ij}^{-1} \tilde{Y}_{ij} \quad (3.13)$$

$$C = \begin{matrix} \text{Tracks} \\ \left[\begin{array}{cccccc} c_{11} & c_{12} & \bullet & \bullet & \bullet & c_{1N} \\ c_{21} & c_{22} & \bullet & \bullet & \bullet & c_{2N} \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ c_{N1} & c_{N2} & \bullet & \bullet & \bullet & c_{NN} \end{array} \right] \end{matrix} \left. \vphantom{\begin{matrix} \text{Tracks} \\ \left[\begin{array}{cccccc} \end{array} \right] \end{matrix}} \right\} \text{Obs}$$

$$c_{ij} = \begin{cases} \text{Arbitrary large number if } m_{ij} \text{ is } 0 \\ d_{ij}^2 \text{ if } m_{ij} \text{ is } 1 \end{cases}$$

Equation (3.12) can be written in its matrix form and simplified down to

$$S_{ij} = \begin{bmatrix} p_{11}^- + r_{11} & p_{13}^- \\ p_{31}^- & p_{33}^- + r_{22} \end{bmatrix} \quad (3.14)$$

Using equations (3.6), (3.13) and (3.14), d_{ij}^2 is calculated as follows.

$$d_{ij}^2 = \frac{\begin{bmatrix} \tilde{y}_{i11} & \tilde{y}_{i21} \end{bmatrix} \begin{bmatrix} p_{33}^- + r_{22} & -p_{13}^- \\ -p_{31}^- & p_{11}^- + r_{11} \end{bmatrix} \begin{bmatrix} \tilde{y}_{i11} \\ \tilde{y}_{i21} \end{bmatrix}}{\left((p_{11}^- + r_{11}) * (p_{33}^- + r_{22}) - p_{13}^- * p_{31}^- \right)}$$

Recall here that $\tilde{y}_{i11} = \tilde{y}_{id}$ and $\tilde{y}_{i21} = \tilde{y}_{i\theta}$.

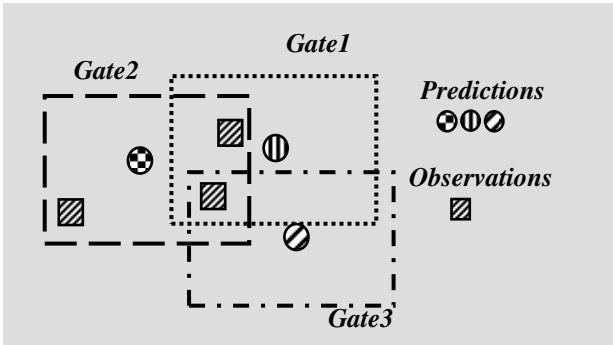


Figure 5: Conflict situation in data association

The cost matrix demonstrates a conflict situation where several observations are potential candidates to be associated with a particular prediction and vice versa. A conflict situation is illustrated in figure 5.

To resolve the conflicts, the cost matrix is passed on to the “Assignment Solver” block which treats it as the well

known assignment problem.

G. Assignment Solver

The assignment problem is stated as follows.

Given a cost matrix of elements C_{ij} , find a matrix $X = \{x_{ij}\}$, such that $C = \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij}$ is minimized

$$\text{subject to } \begin{cases} \sum_i x_{ij} = 1 \forall j \\ \sum_j x_{ij} = 1 \forall i \end{cases}$$

Here x_{ij} is a binary variable used for ensuring that an observation is associated with one and only one track and a track is associated with one and only one observation. This requires x_{ij} to be either 0 or 1 i.e. $x_{ij} \in \{0,1\}$.

There are several algorithms for finding matrix X. The most commonly used among them are Munkres algorithm [9] and Auction algorithm. We use the former in our application. Matrix X below shows a result of the “Assignment Solver” for a 4x4 cost matrix. It shows that observation 1 is to be paired with prediction 3, observation 2 with prediction 1 and so forth.

$$X = \begin{matrix} \text{Predictions (Tracks)} \\ \left[\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{array} \right] \end{matrix} \left. \vphantom{\left[\begin{array}{cccc} \end{array} \right]} \right\} \text{Observations}$$

These pairs are then finally passed on to the relevant Kalman filters to estimate the states of the concerned targets.

All the steps “A” through “G” are repeated for ever in the loop in figure 1. However, there are certain cases where some additional steps have to be taken too. These steps and the circumstances where they become relevant are explained in the next section. Together these three steps are called “Track Maintenance”.

H. Track Maintenance

In real conditions there would be one or more targets that are detected in the current scan but did not exist in the previous scans. On the other hand there would be situations where one or more of the targets being tracked would no more be in the radar FOV. The first case is the “New target Identification” whereas the latter one is the “Observation-less Gate Identification” case. A new target is identified when its observation fails all the already established gates i.e. when all the elements of a row in the “Gate Mask” matrix M are zero. The “new target identifier” starts a counter for the newly identified target. If the counter reaches 3 in five scans, the target is confirmed and a new track is initiated for it. The

counter is reset every five scans. The case of “*Observation-less Gate*” indicates the disappearance of a target from radar FOV. This is manifested when all the elements of a column in the “*Gate Mask*” matrix M are zero. The “*Obs-less Gate Identifier*” looks for 3 consecutive “*misses*” in 5 scans to confirm the disappearance of a target. The “*Track Init/Del*” initiates or deletes a track when needed.

IV. APPLICATION MAPPING TO MPSOC

We coded the application in ANSI C and distributed the application over different processors as distinct functions. Communication among the function is such that we have to implement a producer-consumer architecture for the system as shown in figure 6. Details of this architecture can be found in [15]. Similar considerations have been proposed in [10, 11, 12].

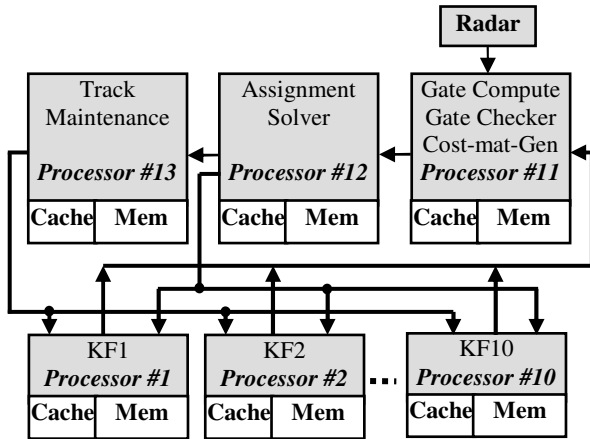


Figure 6 : MPSoC on FPGA

Kalman filter, as mentioned earlier above, is recursive algorithm looping around prediction and correction steps. Both these steps involve matrix operations on floating point quantities. This makes the filter a strong candidate to be mapped onto a separate processor. Thus for tracking 10 targets at a time, we need 10 identical processors executing 10 Kalman filters.

The assignment-solver is an algorithm consisting of six distinct iterative steps [9]. Looping through these steps demands a long execution time hence this function cannot be combined with any of the other functions. So the assignment solver is another strong candidate to be mapped onto a separate processor

The “*Gate Compute*” block regularly passes information to “*Gate Checker*” which in turn, is in constant communication with “*Cost Matrix Generator*”. So we group these three blocks together and map them onto a single processor to minimize inter-processor communication which would have required extra buffers and would have added to the complexity of the system. Avoiding unnecessary inter-processor communication is also desirable for saving power.

The three blocks of the “*Track Maintenance*” sub-function individually don’t demand heavy computational

resources, so we grouped them together for mapping onto a processor.

Using Altera’s Quartus II and SOPC design tools, we implemented the system with the NIOS II processors. The choice of using NIOS II (e) or NIOS II (s) is driven by the results of several tests done on the codes. Inserting time stamps into the code at strategic points, we came to know that the filter spent more than 90% of the time in multiply/divide operations. So the optional “hardware multiply” and “hardware divide” are included with the standard processors for the filters to augment their ALU’s. This augmentation is termed as “*custom instruction*” in NIOS II literature [14]. It is helpful in speeding up the filtering functions because the filters involve numerous multiply/divide operations on floating point numbers.

V. DISCUSSION AND RESULTS

In the very early stages of the work, we implemented a 2-state Kalman filter in hardware using VHDL. It not only took enormous design efforts but also consumed 23,327 LUTs on the FPGA. In fact one 2-state Kalman filter needs about 48% of the Stratix II 2S60 FPGA resources.

In our current design we are using ten 4-state filters apart from the other system components. Implementing a fully hardware system with ten 4-state filters would have been simply infeasible. The NIOS II (e) consumes 600 to 700 logic elements (LEs) whereas NIOS II (s) consumes 1200 to 1400 LEs. The whole MPSoC that we propose consumes almost 30,000 logic elements which are 50% of the reconfigurable resources on Stratix II 2S60 FPGA.

The radar sensor we are using has a PRT of 20 ms so we have to go round the “data association-filtering” loop for all the 10 targets in less than 20 ms. Using 100 MHz clock, the slowest function in the application i.e. the filter takes 15 ms to complete the prediction-correction loop discussed earlier. This is well below the 20 ms threshold set by the radar PRT.

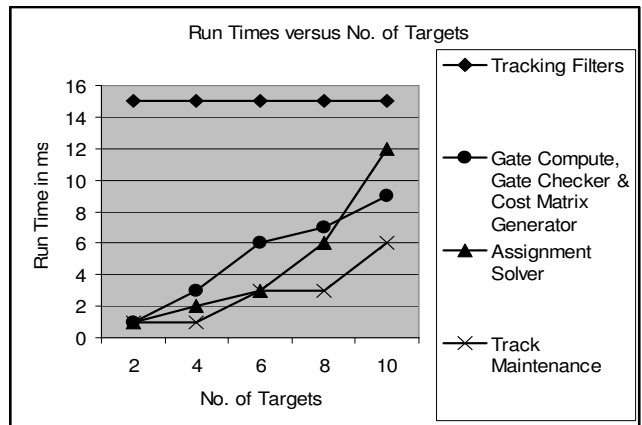


Figure 7: Run times Vs No. of targets

Figure 7 shows how the run times of different functions vary with respect to the maximum number of targets being tracked. Because there are as many filters as the maximum

number of targets and all the filters run concurrently on separate processors, their run time remains unchanged whatever the number of targets and filters. The run time for the combined functions of Gate-Compute, Gate-Checker and Cost-Matrix-Generator varies almost linearly with the number of targets. Here it would be expected that the run time vary exponentially because the number of elements in Gate Mask matrix M and cost matrix C is the square of the number of targets. But a higher number of targets also increase the number of observations failing one or several gates. As discussed above, the computations reduce considerably when an observation does not pass one or several gates. This accounts for the almost linear change in the run time for these functions. The Assignment Solver shows an exponential rise in run time because the algorithm is $O(N^2)$. The Track Maintenance function behaves randomly since the numbers of disappearing and new targets vary randomly.

VI. RELATED WORK

To our understanding, comprehensive literature about implementation of complete MTT system in FPGA does not exist. Some work has been done on isolated parts of the MTT system. For example in [2] the authors propose an FPGA implementation of the estimation part of MTT system. Apart from being limited only to estimation, this is a fully hardware implementation. As mentioned earlier in the introduction, fully hardware designs lack the flexibility and programmability needed for the ever evolving modern day applications. A fully hardware implementation of the Kalman filter only, is also proposed in [3]. This implementation also has all the above mentioned limitations for the same reasons. An attempt to implement an MTT system in hardware for maritime application is documented in [7]. Besides being a completely hardware implementation and specific to maritime applications, the work presented here is inconclusive.

In contrast to the works mentioned above, we consider a complete MTT system implementation. Our MPSoC architecture of the system is inherently flexible, programmable and scalable. Thus it can evolve very easily with advances in technology, improvement in application algorithms and market demands. The use of several concurrently running processors achieves the overall run time constraints. Several low frequency processors running concurrently consume less power compared to a single processor with a high frequency [13]. The reconfigurability of the components enables their customization according to application and further saving FPGA resources.

VII. CONCLUSIONS AND FUTURE WORK

We presented an application-specific MPSoC architecture for MTT based driver assistant system. The system is implemented in FPGA with Altera's SOPC builder and

NIOS II processors. The system uses 50% of the reconfigurable resources on a Stratix II 2s60 FPGA which contains 60,000 logic elements. We currently cater for a maximum of 10 targets. The system is easily evolvable. It is not only a complete embedded MTT solution but is also economical and power efficient as compared to fully hardware implementations. We plan to take the system further up the evolution hierarchy in the near future. On the application side, we intend to take dynamically varying number of targets into account. This would require us to dynamically vary the number of filter configurations on the FPGA. Tracking precision can be improved through better / newer algorithms e.g. extended Kalman Filter (EKF), auction algorithm for assignment, radar-signature aided data association etc.

On the architecture side, new processor configurations and interconnections shall be evaluated for run time and power consumption improvement. To integrate the system with other electronic safety systems onboard a vehicle, we shall add a CAN (Controller Area Network) interface to the system.

REFERENCES

- [1] Samuel Blackman and Robert Popoli, "Design and analysis of modern tracking systems", Artech House Publishers 1999, ISBN 1-58053-006-0.
- [2] Zoran Salcic and Chung-Ruey Lee, "FPGA-Based Adaptive Tracking Estimation Computer" IEEE transactions on aerospace and electronic systems, 2001.
- [3] Salcic, Z. et al. "Scalar-based direct algorithm mapping FPLD implementation of Kalman filter." IEEE Transactions on Aerospace and Electronic Systems, 2000.
- [4] Kalman, R. E. "A New Approach to Linear Filtering and Prediction Problems", Transaction of the ASME--Journal of Basic Engineering, pp.35-45(Mar 1960).
- [5] Welch, G and Bishop, G. 2001. "An introduction to the Kalman Filter", <http://www.cs.unc.edu/~welch/kalman/>
- [6] Brookner, E. "Tracking and Kalman filtering made easy" John Wiley & Sons Inc. 1998.
- [7] Boismenu, Y. "Etude d'une carte de tracking radar---" Thèse de doctorat année 2000, Université de Bourgogne, France.
- [8] P. Konstantinova et al. "A study of Target Tracking Algorithm Using Global Nearest Neighbor Approach" CompSysTech' 2003.
- [9] Munkres' Assignment Algorithm, Modified for Rectangular Matrices <http://csclab.murraystate.edu/bob.pilgrim/445/munkres.html>
- [10] Joost, R.; Salomon, R. "Advantages of FPGA-based multiprocessor systems in industrial applications" Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE Volume, Issue, 6-10 Nov. 2005.
- [11] Epson's Breakthrough REALOID Printer SOC Powered by Multiple Xtensa Processors http://www.tensilica.com/products/lits_success-stories_epson.htm
- [12] Hannu Penttinen, Tapio Koskinen, Marko Hännikäinen."Leon3 MP on Altera FPGA" Altera Innovate Nordic 2007, Final Project Report 8/28/2007
- [13] Frank Schirrmeister Imperas, Inc." Multi-core Processors: Fundamentals, Trends, and Challenges" Embedded Systems Conference 2007 ESC351.
- [14] Altera Corporation. "NIOS II processor reference handbook". www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf
- [15] J. Khan et al. "An MPSoC Architecture for the Multiple Target Tracking Application in Driver Assistant System" 19th IEEE International Conference ASAP08, 2-4 July 2008, Leuven Belgium.