Formal Models for Cognitive Systems

Arunkumar Ramaswamy*† and Bruno Monsuez* and Adriana Tapus*
*Department of Computer and System Engineering,
ENSTA-ParisTech, 828 Blvd Marechaux, Palaiseau, France
†VeDeCom Institute, 77 rue des Chantiers, 78000 Versailles, France
Email:{arun-kumar.ramaswamy; bruno.monsuez; adriana.tapus}@ensta-paristech.fr

***Abstract*—A robotic system is an integration of various systems such as perception, navigation, planner, controller, etc. The adaptation of the robotic system to the dynamic environments is embedded in the functionality of the constituent systems. This severely limits the configuration space of such systems. The modeling of the variability in the solution space can expand this design space, help finding the best possible solution, and perform run-time adaptation of the system. In this paper, a formal specification using primitive compositional elements is proposed to analyze the solution space and to streamline the decision making using non-functional properties of the system. The relevancy of the model is demonstrated using a case study on a vehicle tracking problem.**

I. Introduction

In recent years, Component Based Software Engineering (CBSE) is proposed in Robotics to manage complexity, reusability, and portability. In CBSE, components are standalone systems having predefined interfaces, which perform a specific functionality. Typically, these components use a communication middleware to interact with each other and will not impose any specific architecture on the designer. Application engineers consume these components from the component library and compose them to build a more complex system so as to achieve a unique purpose [1]. The final system can be viewed as Systems of Systems (SoS) with components as heterogeneous independently operable systems. The intelligence resides in these individual functional systems and no significant effort is invested during initial stages in addressing the compositional issues of these systems. The problem gets further aggravated when these components are used in safety critical systems, such as autonomous driving in automotive domain. In such complex systems, the emergent behaviors caused by the interaction of different components supplied by different vendors cannot be overseen without a model-based approach. This is identical to the issue in robotics where heterogeneous systems developed separately operate together under strict real-time requirements interacting with the physical world.

The solution space for implementing a functionality in robotics domain is large. Developers anticipate and assume several facets about the target environments while implementing and testing software components. For example, a developer decides to use a particular localization algorithm that requires the sensor data to have a certain confidence level and resolution, which may not be satisfied in the final system composition. The underlying reason is that no formal model is used to analyze and manage the solution space available to various stakeholders. Such a model will help in - system level reasoning, making tradeoffs, comparing decisions, and to formally proving and validating the final implementation. However, for various reasons described below, system designers cannot arrive at a model that can be analyzed during initial stages of development: a) Most of the robotics software components execute under tight temporal constraints. The timing properties of components cannot be predicted correctly without the knowledge of the target platform; b) The decision on which algorithm to choose is decided by the domain expert during the implementation stage. Hence, the designer do not know anything about the operational profile of those functionalities and its prerequisites, dependence, potential interactions etc.; c) Most of the functionalities, such as localization, perception, and controllers are developed parallely by domain experts under the assumption that the data they produce and consume are interpreted correctly. In addition, only the functional verification of such individual modules are conducted and the performance of these modules cannot be guaranteed when it coexists with other systems; d) Robotics domain is not mature enough to accept any standards and ontology, similar to aerospace and automotive domains, which helps in knowledge representation and sharing.

Further, although futuristic, Artificial Intelligence can be applied to the system design, including selecting the appropriate components from the component library, composing them, and verifying. Given a functionality, a middleware, and a target platform, the system can pick components functionality wise, analyze its operational profile, and deduce the timing properties against the hardware platform, compose them in such a way to meet the non-functional properties, such as timing response. For example, when a path planning system with specific middleware, target platform, and required quality are specified, the system can automatically propose the component composition. As a first step towards addressing such problems, different aspects of the design space must be formally specified. It can help to quantify and streamline the thought process in making decisions and tradeoffs during system design. In this paper, we present a formalism to specify components using their functional properties and provide basic composition elements to reason about the solution space against performance objectives of the system.

The paper is organized as follows: Section II provides related works. An overview of the current trends in software development in robotics is given in section III. Section IV provides an example that shows the issues association with 'black boxed' component based approach that exist in robotics. Requirements for a robotic component is discussed in section V and basic compositional elements that can be used to formally model the system is proposed in section VI. Section VII demonstrates the use of formal models with the help of a case study on vehicle tracking problem. The results and key observations are discussed in section IX.

978-1-4799-2722-7/13/$31.00

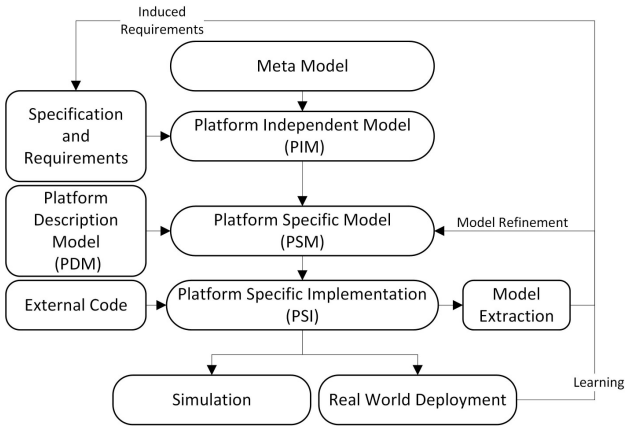


Fig. 1: Model Driven Engineering in Robotics

II. Related Works

The focus in various stages of developmental cycle for a robotic system can be broadly classified into three parts - problem space, solution space, and operational space. The initial work on robot architecture began with two extreme approaches - one is based on sense-plan-act paradigm [2], which is a deliberative approach and on the other end is this purely reactive Brooks' subsumption architecture [3]. To take advantage of these two extreme approaches, a number of hybrid architectures were then proposed. The most popular one is Gat's three layer architecture that uses controller, sequencer, and deliberator layers to enable the robot to make high level plans and at the same time reactive [4]. The hybrid architectures concentrated mainly on how low level reactive behaviors can be coordinated using high level planning and decisional algorithms. Following this, Claraty [5] and LAAS architecture [6] were proposed. They use a layered approach and provide tools to coordinate low level functional elements using higher decisional layers. All these architectures have been concentrated only on the operational space of the robot.

More recently, advanced software engineering concepts are being incorporated in architecting robotic systems. The Smartsoft framework is based on a model driven toolchain that provides the model transformations and code generation steps for service robots [7]. The European project on Best Practices in Robotics (BRICS) provides guidelines and a framework to develop robotic components [8]. They are based on the separation of concerns between the development aspects of Computation, Communication, Coordination, Configuration, and Composition. However, no formal models have been specified in these frameworks.

For many reasons, to address such issues, roboticists have concentrated mainly only on adaptivity at the functionality level in operating space. They do so by incorporating intelligence in individual systems, such as planners, collision detectors, navigation, controllers, etc., to address this uncertainty. However, none of these architecture tap the capability in solution space explicitly. This severely limits the space in which the self-adaptability can be applied. In our approach, we formally include multiple realizations of same functionality with different non-functional properties in the system design. The advantage is of two fold - a best possible system composition from the solution pool can be selected for a given application and variation points can be explicitly included where decision is made during run-time depending on the context. But in most of the component based frameworks proposed in Robotics, there is an explicit supervisory unit. In some frameworks it is called director, some refer it as coordinator some other as component manager. But lack of formal models restrict the scope of the manager to a larger extend.

III. Software Development in Robotics

The current trend in software development in robotics is model centric approach. Model Driven Engineering (MDE) is recommended for complex systems for ensuring reliability and robustness. The most common MDE approach in robotics starts with a Platform Independent Model (PIM) that conforms to a domain metamodel and that satifies the requirements. The level of abstraction of the model is gradually reducing by adding platform spedific details. A Platform Description Model (PDM) contains the hardware platform details, middleware specific information, operating system etc. Using PDM and PIM, a Platform Specific Model is generated with hot spots in which the developer can insert "hand coded" functionalities. However, a lack of common ontology and standard semantics in robotics restricts to a larger extend the use of models in software development process.

The model driven work flow cannot directly be applied in the robotics domain. The primary feature that distinguish a robotic system with respect to other embedded systems is that it operates in a highly dynamic environment. This unpredictability spans over various phases in software development - from requirement specification, system design, implementation, integration and till it is deployed in real world scenarios. The system cannot be realized in an unidirectional process flow because the solution for a robotic problem cannot be finalized during the design time. It is because neither the problem space nor the target environment cannot be completely modeled as in embedded systems. Hence, the current trend is to generate a container with slots that allows the developer to insert the "hand coded" functionalities. However, such approach may result in inconsistencies with models since most of the code in robotics domain is through external class libraries, which does not follow any standard semantics. This is commonly termed as *round trip problem* in software engineering literature. Hence a more iterative approach is required in the process. Figure 1 shows a model centric process flow that can be adopted in robotics. The refinements of the model happen at various stages of the design flow and such an approach can also help in a certain degree of agile developmental approach. It is a cyclical process to find the best possible subset of solutions for a given problem. It will not essentially be a single solution because certain algorithms can only be selected during the run-time process as a function of the context. Hence, such an approach can be easily extended to introduce run-time variability in the system. In this paper, we use the term system and component interchangeably to indicate a SoS perspective. In higher layers the term system is used while in lower layers, the term component is used.

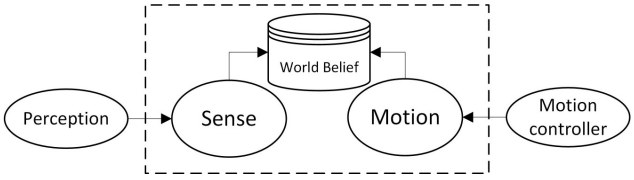


Fig. 2: Localization System

IV. Motivating Example

Consider a system implementing a localization function for a mobile robot. The localization algorithm uses the information about the environment from the perception system and vehicle's motion from motion controller to estimate its pose with respect to a known map. The system comprises of mainly two components and a database representing its world belief as shown in Figure 2. The sense component uses Bayes rule to update the belief using the current observation and measurement probability. The motion component updates the effect of the vehicle motion on its belief. In general information is gained by the sense component and it is lost during the vehicle motion. By a series of sense and motion update functions on the belief, the confidence on the vehicle's pose is increased.

In real time scenario, the vehicle will not exhibit move-stop behavior while executing motion and sense components. The vehicle will be continuously moving and perception system will be continuously sending the information. Hence, the sense and move component almost operate parallely. The implementer of the system assumes that the most recent data are available on time. But in real-time operation many anomalies can occur. The perception component may operate in a lower frequency because of some sensor breakdown and in which case the system may provide wrong estimate of its pose. If the localization system is used in applications such as autonomous driving in real world scenarios, the outcome will be disastrous. The main drawback in component based approach is that the space for a component developer to mitigate the faulty situation is limited. The design space is within the required and given interfaces and the component is unaware of the system's objectives. For example this approach will rule out the following alternatives: 1) It can request to the motion controller to slow down its motion to cope up with the lower operational frequency of perception component. This is an example to lowering the functionality to an acceptable level and still achieving the system's objective. 2) It can request the perception component to provide information at a lower confidence level but at a higher frequency. 2) It can provide an emergency stop alarm to other components in way to mitigate the problem when the confidence level reduces to unacceptable levels. But these alternatives can be possible only if a high level system model can be generated from the component composition and making the component aware of the system composition. This is a paradoxical situation because component based development use black box approach to reduce complexity and increase reusability. The root cause of such conflicts is because component based approaches are designed in the operational space. The optimization and dynamism is happening only in this space neglecting the problem space and solution space on top of it. By incorporating these spaces, it expands the adaptability of the components enormously. In this paper, a model is proposed in solution space, which connects the problem space above and the operational space below. The performance objectives of the system derived from the problem space and the abstract non-functional properties derived from the operation space are correlated in the solution space to find the best possible subset of solutions. It can also guide the iterative developmental approach mentioned in section III.

V. Requirements for a Robotic Software Component

1) Composability: A component is said to be composable if its core properties do not change upon integration. In other words, the composability of component guarantees preservation of its properties across integration with other components. A highly composable component can freely move around in the design space and assemble itself to a more complex structure that is semantically correct.

2) Compositionality: Compositionality allows to deduce properties of a composite component from its constituent components. It enables hierarchical composition of components and provides correctness of the structure. The reusability of component is enhanced by providing compile-time guarantees by verifying that formal parameters and actual parameters match regardless of the software module's location. The behavior of the reused components can be predicted in the new configuration and the result of the composition can be analyzed for anomalies.

3) Static and Dynamic Variability: Static variability is the configuration of the system and dynamic variability is related to the coordination of the components. Configuration defines which system can communicate each other and coordination determines when such communication can occur. Configuration can be completely specified during design time while coordination is achieved by allowing variability during design time and run-time dynamic invocation.

4) Component abstraction: Separating component specification and implementation by abstraction makes the system extensible, portable, and reusable. Component abstractions standardize functionalities, interfaces, communication mechanisms, timing behavior, etc. The level of abstraction may vary with process stages and across developmental cycles. Initially the abstractions will contain basic functionalities and later as the design progresses extra specifications on non-functionalities, such as performance, timing behavior can be added.

5) Technology Neutrality: The component properties and specification should not depend on a specific technology. Software technology neutrality and hardware neutrality to some extend is achieved by many of the already available frameworks, such as, ROS [9], Player project [10], etc.

VI. Formal Elements to Represent Component composition

This section proposes three atomic elements to formally represent the composition of systems as shown in Figure 3. Using the combination of logical dispatch gates, ports, and connectors various computation logic that are relevant in robotics can be realized.

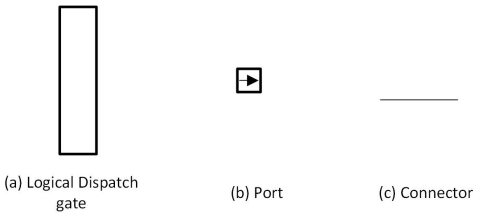


Fig. 3: Compositional elements

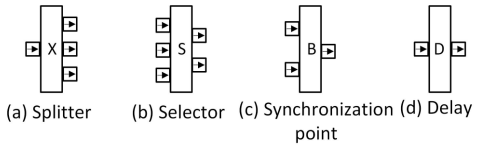


Fig. 4: Example composite elements

The logical dispatch gates represent basic logical operations such as, multiplexing, selecting, etc. The data enters or leaves through the ports from the gates and each port has a particular abstract data type associated with it. The port can be in port or out port depending on whether the data enters or leaves the gate. They also contain basic data conversion computations. The computations in gates and ports are instantaneous actions and thus does not consume time. The computation intensive actions are represented using connectors. The connectors connect two semantically compatible ports. The connectors are hierarchical in the sense that it can represent another system with similar compositional elements.

Figure 4 shows some example composite elements that are realized using the three atomic elements.

a) Splitter creates n copies of the input data. This is realized using gate with n out ports and 1 in port.

b) Selector selects m out of n input data. A selector gate is comprised of m out ports and n in ports.

c) Synchronization Buffer act as a synchronization point between data elements.

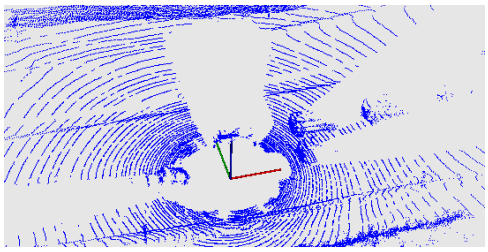
d) Delay passes the data in the input port after a time delay.

VII. Case Study: Vehicle Tracking System

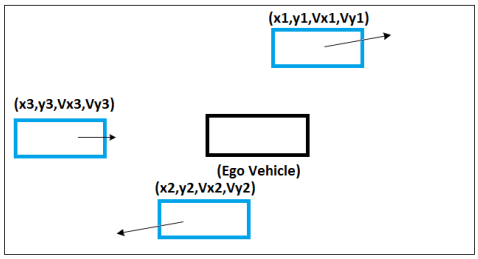
The design of a vehicle tracking system is used as a case study to explain the formal representation of solution space. Section VII-A provides problem description with general specification of the system. The anticipated use-cases are listed in section VII-B. This is essential for a SOS approach in which the system's performance can be tailored with respect to the target application. The main intention of assuming the usecases is to find the various performance objectives of the system at an earlier stage and the list need not be complete. System block diagram is described in section VII-C and the abstract data types are defined in section VII-D. Formal representation using compositional elements of segmentation system is shown in section VI.

A. Problem Description

The system should detect vehicles in the environment and compute its state. The state of a vehicle consists of its 2D position and velocity. The system receives point cloud input from a high definition 3D Lidar, as shown in Figure 5a. Figure 5b shows a sample visualization of the output as 2D map of the environment with blue rectangles indicating the tracked vehicles and arrows indicating the heading direction.



(a) Point Cloud input



(b) Sample output

Fig. 5: Vehicle Tracking System

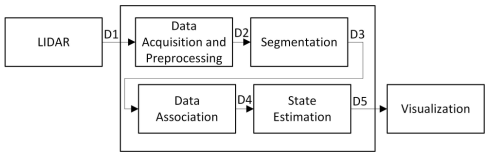
B. Anticipated Use-cases

The following is a list of use-cases in which the tracking system might be used. The key performance parameters are also indicated for each use-case.

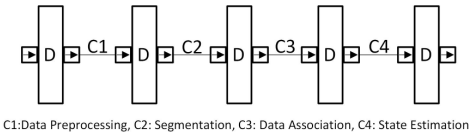
1) Ground Truth Generation: Since a high performance lidar system is used for tracking, the system can be used for generating ground truth for benchmarking and evaluating other systems. This application requires a higher confidence level and high resolution data.
2) A path planning system of a autonomous vehicle can use it to detect obstacles and dynamically plan its path. The tracking should provide minimum confidence threshold and resolution less than 10cm and acceptable timing response.
3) It can be used in a traffic detection system to detect the speed of vehicles in a high way lane. In this case, the required vehicle state parameters are the speed of the vehicles.
4) A parking lot can use it to count the number of vehicles entering and leaving a given check point. This application does not need the vehicle velocity, but it only requires location of the vehicle to count the number of vehicles passing by.
5) A mapping system can use it to generate the occupancy map of the environment. The resolution of the position of the tracked vehicle determines the resolution of the map.

C. System Block Diagram

The classical method used for tracking applications is shown as block diagram in Figure 6a. Description of each



(a) Functional View



(b) Formal Model

Fig. 6: System View

block is given below:

1) Lidar: Velodyne HDL-64E is used for acquiring point cloud data. It is a high definition laser scanning system with a rotating sensor head containing 64 semiconductor lasers and can generate approximately 1.33 million points per second. A serial interface is used to vary the rotation speed from 5Hz to 20Hz and to configure other parameters. The sensor broadcast UDP packets with range information through a high speed ethernet interface. Each packet contains range and angle data of 12 laser firings with GPS timestamp. The timestamp of each point is computed by subtracting an offset from the timestamp.

2) Data Acquisition and Preprocessing: The raw UDP packet data is converted to point cloud frames in data acquisition block. A point cloud is a set of 3D points with a fixed point of reference. The data acquisition block send a point cloud frame for every 360 degree rotation of the sensor.

3) Segmentation: It is a clustering process that converts the unorganized cloud into smaller components so that points with similar attributes are grouped and thus facilitating in further data analysis. The input to this block is point cloud frame and it outputs labeled point cloud so that all the points corresponding to same object are given the same label.

4) Data Association: In this block, the objects found in each frames are associated using some features. It receives segmented point cloud and outputs the position of each object in a frame and its associated label across frames. For example, an object with label 'L1' in two different frames will represent the same object.

5) State Estimation: The velocity of the vehicle is computed by Bayesian based filtering process. The position of the vehicle in different frame is obtained after data association step. It outputs the estimated state of the vehicle - pose and velocity.

The formal model for the system view is shown in Figure 6b. Here the different blocks of functional view in Figure 6a is separated by a delay element. The computation in each block is represented by the four connectors - C1, C2, C3 and C4.

D. Abstract Data Type Definition

Abstract data types associated with each port in the model in Figure 6b is detailed in Table I. Cx_{in} and Cx_{out} represents the abstract data type associated with the port connected to the input and output of the connector Cx respectively. In this data type representation, the functional and non-functional attributes associated with each data type are separated in the below notation.

$$\langle Label : [Func.\ Attributes]\ (Non - Func.\ Attributes)\rangle$$

The attributes are indicated by comma separated labels and array attributes are indicated using a notation $\langle type\rangle\ (number)$, for example $\langle x, y\rangle\ (10)$ represents an array of (x, y) of size 10.

The non-functional attributes can be deduced from the following ways

- Performance parameters derived from the anticipated use-cases.
- Parameters determining Quality of Service (QoS) of the system.
- Abstract properties for the family of hardware components. For example, resolution property for image acquisition devices, scan angle for laser ranging devices, etc.

By analyzing the anticipated use-cases listed in section VII-B we can deduce three parameters - computation time required, resolution and confidence. These three performance parameters are used in this case study as non-functional attributes of the system.

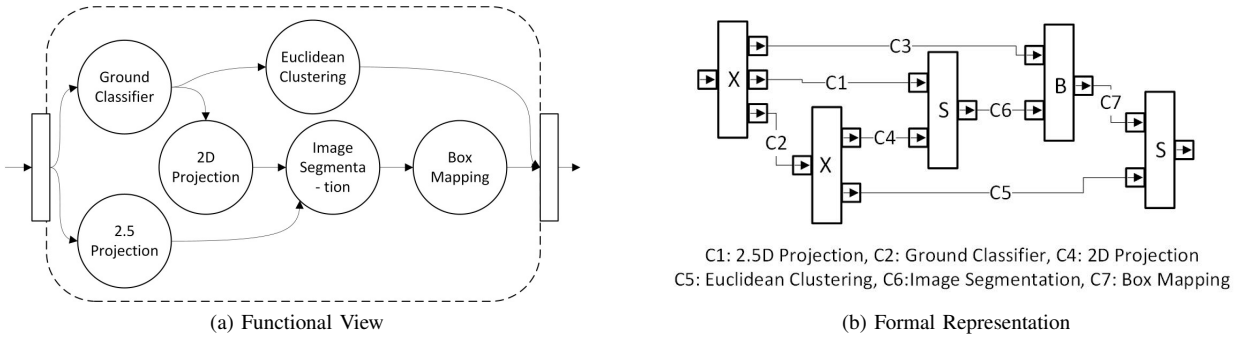
TABLE I: Datatype Definition

Label	Datatype
$C1_{in}$	$\langle Vel\_Data : [\langle r, \theta_r, \theta_v, I\rangle\ (12)\ ; \langle t_{aq}\rangle\ (1)]\ ()\rangle$
$C1_{out}, C2_{in}$	$\langle PC\_Frame : [\langle x, y, z, I\rangle\ (N_{fr})\ ; \langle t_{fr}\rangle\ (1)]\ (t_{cp}, Conf, Res)\rangle$
$C2_{out}, C3_{in}$	$\langle PC\_Seg : [\langle x, y, z, I, L_{ob}\rangle\ (N_{fr})\ ; \langle t_{fr}\rangle\ (1)]\ (t_{cp}, Conf, Res)\rangle$
$C3_{out}, C4_{in}$	$\langle Obj\_pos : [\langle x, y, L_{ob}\rangle\ (N_{ob})\ ; \langle t_{fr}\rangle\ (1)]\ (t_{cp}, Conf, Res)\rangle$
$C4_{out}$	$\langle Obj\_state : [\langle x, y, V_x, V_y, L_{ob}\rangle\ (N_{ob})\ ; \langle t_{fr}\rangle\ (1)]\ (t_{cp}, Conf, Res)\rangle$
Label Description-	
Functional Attributes- r:range, θ_r:rotation angle, θ_v:vertical angle, (x,y,z):3D position, I:intensity, t_{aq}: time of packet acquisition, t_{fr}: timestamp of the frame N_{fr}: No. of points in a frame, L_{ob}: Label of an object, N_{ob}: No. of objects (V_x, V_y): x and y component of velocity	
Non-Functional Attributes- t_{cp}: computation time required from acquisition of the frame, $Conf$: Confidence, Res: Resolution	

E. Segmentation

Due to space constraints, only segmentation block is considered for formally specifying the solution space. Large number of segmentation algorithms are available to segment point cloud data. Figure 7a shows a functional view of the subset of those algorithms under consideration. The circular nodes represent a specific algorithm and the connecting arrows indicate the flow of data. In general, a point cloud data can be segmented into clusters using two methods. 1) By directly processing the 3D data and clustering using some criteria such as Euclidean distance or 2) By mapping the point cloud to a 2D image and performing segmentation in 2D space using image processing techniques and then projecting back to the 3D space to compute the final point cloud clusters. A brief description of each algorithm is given below.

Ground Classifier: The algorithm can classify ground and non-ground points from a point cloud frame [11]. By rejecting



(a) Functional View

(b) Formal Representation

Fig. 7: Segmentation System

the ground points, this algorithm can significantly reduce the number of points for further processing. The input to this algorithm is the point cloud frame and the output is the point cloud corresponding to non-ground points.

2D Projection Method: This method uses (x,y) coordinates of the point cloud and converts it to a two-dimensional binary image. This is best suited for point clouds consisting of non-ground points and hence, it receives data from a ground classifier node.

Euclidean Clustering: This algorithm uses Euclidean distance to cluster the point clouds [12]. It can be applied on non-ground point cloud data to generate the final point cloud clusters.

2.5D Projection: This algorithm can convert a point cloud to a 2D grayscale image [11]. Ground points are automatically rejected and there is no requirement for explicitly classifying the ground and non-ground points. The input to this node is a point cloud frame and outputs the grayscale image.

Image Segmentation: Using image processing techniques, connected regions are found in the image and a bounding box is computed for each region. The input to this node should be a 2D binary or grayscale image. Hence, it can either use the data after 2D projection algorithm or data after 2.5D projection. The results of this node are a list of 2D bounding boxes.

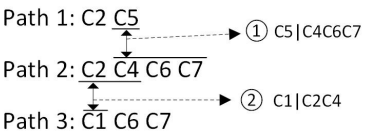
2D to 3D Box Mapping This computation block receives 2D boxes and the correspondence point clouds and generates 3D point cloud clusters.

Formal model for the segmentation system is shown in Figure 7b. The model is at the hierarchically lower level of model shown in Figure 6b. In fact the segmentation model is the expanded version of the connector C2 in graphical model in Figure 7b. It is to be noted that the connector C2 in Figure 7b and 6b represent different computations since they are in different hierarchical levels.

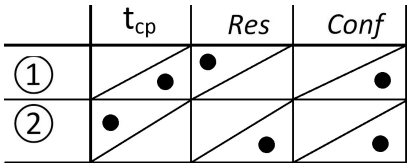
1) Path Tracing and Variability Analysis: The formal graphical view of the segmentation component using the compositional elements listed in section VI is shown in Figure 7b. The graph shows that the required functional result can be achieved in multiple sequence of algorithms. However, the non-functional properties such as computational time, confidence, resolution vary for different sequence of algorithms. To compare these heterogeneous algorithms with respect to the performance objectives of the component is difficult. For example, comparing 6 different computations against 3 performance objectives requires at least 18 comparisons so as to decide the best possible path. However, the decision space can be reduced by computing the variation points and by comparing only the variation points against performance objectives. A variation point is a set of paths that can be plugged in different execution paths and still have the same functional result. The process for analyzing the possible paths and variation points is described below.

The first step is to trace all the possible paths in the solution graph. Figure 8a lists the three paths that provide the same functional output, the point cloud clusters in this case. The paths are represented by a sequence of labels that represent the connectors in that execution path. The paths - 1,2 and paths 2,3 have common sub-paths are indicated in the Figure 8a. The variation point between path 1 and 2 is $C5|C4C6C7$ and between path 2 and 3 is $C1|C2C4$. Now, the decision has to be taken only by regarding these two variation points. There are various methods in which the variation points can be compared against performance objectives. Non-functional properties, such as timing constraints can only be numerically compared after providing the implementation and target platform specifications in subsequent iterations of the round trip methodology described in section 1. But qualitative comparison can be made during design time and pre-implementation stages using methods, such as domain expertise, experience, operation profile analysis etc.

Table 8b compares the two variation points against the three performance objectives identified section VII-B - computation time, resolution and confidence. Each cell in table is divided into two halves. The dot in the upper half indicate that the first sub-path in the variation point is better than the second sub-path with respect to the performance objective corresponding to that column in the table. In first variation point the node C5 competes with the node sequence C4, C6, C7. Functionally it implies that, the sequence of 2D Projection, Image Segmentation and Box Mapping can be replaced by Euclidean projection method. The Euclidean approach directly operates on 3D point cloud and uses Euclidean distance to cluster and hence the time taken is more that of the alternate sequence in the which the point cloud is converted to a 2D image and then clustering is applied in 2D space and then mapped backed into the 3D space. The dot in lower half of the cell corresponding to variation point 1 and objective t_{cp} indicates such comparison. Similarly, the resolution is high for Euclidean approach while confidence is high for 2D image based approaches. This is indicated in the rest of the cells for the first variation point. The second variation point is



(a) Path Tracing



(b) Variability Analysis

Fig. 8: Path Analysis

$C1|C2C4$, in which the 2.5D Projection method can be used as a replacement to Ground classifier and 2D Projection method. Ground point classification is a time consuming approach as compared to 2.5D projection method. Similar analysis were used to fill in the cells accordingly. This approach, although depend on the expertise of the designer and this can be applied during the design time and at the later stage when implementation and platforms specification are added, can be used for more rigorous numerical comparisons.

VIII. Discussions and Results

This section provides results and explains the rationale behind various design decisions taken during different iterations of the development process. By using formal models these decisions can be automated and can guide the component manager to coordinate various computation of the system. In this paper, we considered only the segmentation component for algorithm specific details. The initial use case was the generation of the ground truth and the performance objectives were higher resolution and confidence levels (see Section VII-B). This decision helps to find the best possible solution with a high confidence level. Variability analysis as described in section VII-E1 shows that the two variation points indicate that path 2 provides higher confidence level. Hence, the algorithms in path 2, such as Ground classifier, 2D Projection, Image Segmentation, and 3D Back projection were first developed during the implementation phase. The results after ground classifier and object segmentation are shown in Figure 10a and 10b, respectively.

Figure 9 shows the formal model of the solution space after certain algorithms were implemented and dependencies were obtained. The algorithms associated with the connectors are specified in the Figure 9. Some of the key observations that were made during the process are given below.

1) Alternate Path Representation: In the initial design diagram as presented in Figure 6, the required interface of segmentation system is a labeled point cloud, which indicates the object level clusters in the frame. These clusters were required by the subsequent data associated step to associate the objects across frames. However, there are different category of methods called Random Finite Sets, such as PHD filters, which does not require explicit data association steps. The required input for these kind of filters are the centroid of the point cloud cluster and does not require explicit spatial orientation of the points in the cluster. Hence, the 2D bounding rectangles from the segmentation system can be bypassed to the PHD filtering node without performing data association. The is shown in Figure 9 as C2 connector in the tracking block from segmentation component to the PHD Filter. The classification is performed because there are methods that can operate on the bounding boxes alone that are obtained after segmentation. For example, the bounding box dimensions can be used to classify between a vehicle and a pedestrian without using heavy computation on the point clouds.

2) Induced Requirements: During the development process, the software developer for feature detector component had computed surface normals for point cloud clusters to compute the Point Feature Histograms (PFH) as feature descriptors [13]. PFH features heavily rely on the spatial position of the points in the clusters and for reasonable results, there should not be any distortion in the point cloud. The component was verified using the sample cluster that was obtained when there was no relative motion between the sensor and objects in the scene. However, due to the scan latency of the Velodyne sensor and the motion of the ego vehicle, each point in the point cloud frame will be having its own origin and hence, the point cloud will be distorted. To remove the distortion, there should be an undistortion component in the data pre-processing system. In order to correct this distortion, the undistorter component requires an accurate pose estimator for every point in the point cloud to compensate for the vehicle motion. This in turn requires a high precision Inertial Measurement Unit (IMU) to estimate the pose of the vehicle. These incompatibilities can be automatically captured by the component manager because the non-functional properties, such as quality and confidence are transferred along with the data. So the anomaly of not including an undistorter component can be easily captured by the component manager since the confidence in the data will be low in that case.

3) Impact factors: Selecting the path in the solution space using non-functional requirement alone will not give the best possible path in all cases. There are factors other than technical that steer the design decisions, such as cost and implementation time, etc. For example, the critical component in the scenario explained in section VIII-2 is the feature detector using PFH whose induced requirement after bouncing in several components necessitated to use a very expensive inertial measurement unit. Hence, the cost of using that algorithm is high. In another case, PHD filter mentioned in section VIII-1 is hard to implement and therefore, the impact factor of that component is also high especially if there is less implementation time.

4) Computation Redundancy: The main intention of finding alternate paths as in section VIII-1 is to reduce the number of computations in the solution path in order to reduce the time. This can be obtained even after a component is implemented. Several model extraction techniques can be applied to extract functional models from the user defined code and external class libraries. While analyzing the models that were extracted from certain components, several instances were noticed where some computational nodes can be comfortably skipped or bypassed. For example, there were some nodes in the image segmentation component where image binarization was applied on the input image where the image was already binary.

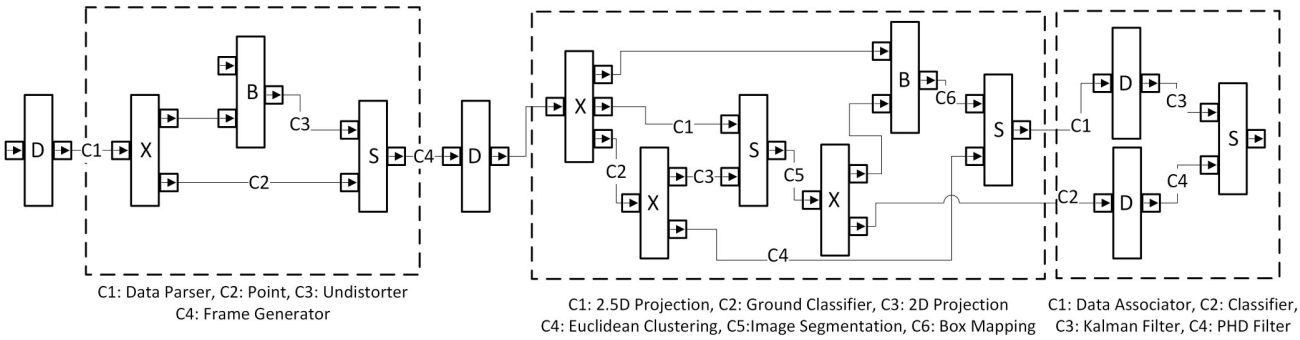
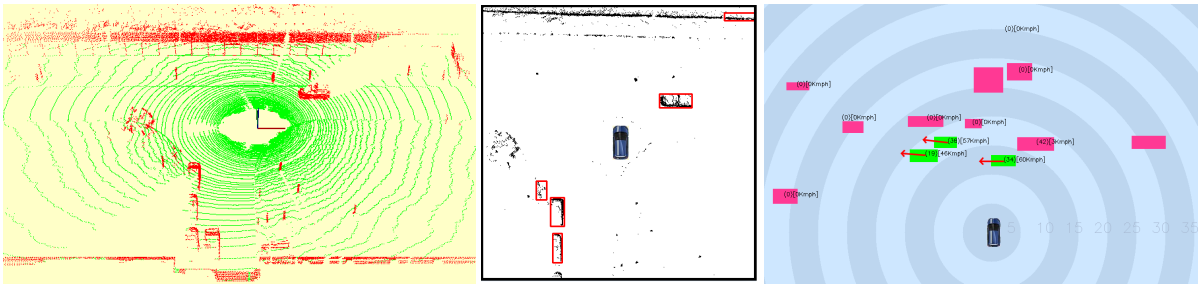


Fig. 9: Formal model for vehicle tracking problem



(a) Ground Classifier (b) Object Segmentation (c) Vehicle Tracking

Fig. 10: Results

IX. Conclusion and Future Work

The observations and results of the case study show that by analyzing the robotic components by using only the functionality, severely limits the space for adaptability. The real differentiators are the non-functional properties that can quantitatively compare various execution paths and can make decisions in the large solution space that exists in robotics. Critical path and Variation point analysis helps to streamline the decision process and to use the domain expertise to trade-off various performance objectives.

The major constraint in robotics so as to adapt formal specification is that there is no standard semantic knowledge that is accepted by the research community. This limits the development of Domain Specific Languages (DSL) and thereby restricts the components to have a formal model that helps in automatic code generation and to provide formal proofs. The next logical step is to analyze the commonly used algorithms in robotics and to extract the high level non-functional properties and attributes and to provide a formal component model for the functional algorithms.

Acknowledgment

This research is funded by VeDeCoM Institute, a French automotive cluster on mobility research. The vehicle tracking experiment were conducted with the help of Ibanez Guzman Javier at Renault S.A.S, France.

References

[1] D. De Laurentis, C. Dickerson, M. DiMario, P. Gartz, M. M. Jamshidi, S. Nahavandi, A. P. Sage, E. B. Sloane, and D. R. Walker, "A case for an international consortium on system-of-systems engineering," *Systems Journal, IEEE*, vol. 1, no. 1, pp. 68–73, 2007.

[2] P. E. Agre and D. Chapman, "What are plans for?" *Robotics and autonomous systems*, vol. 6, no. 1, pp. 17–34, 1990.

[3] R. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14–23, 1986.

[4] E. Gat *et al.*, "On three-layer architectures," *Artificial intelligence and mobile robots: case studies of successful robot systems*, vol. 195, 1998.

[5] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The claraty architecture for robotic autonomy," in *Aerospace Conference, 2001, IEEE Proceedings.*, vol. 1. IEEE, 2001, pp. 1–121.

[6] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, "An architecture for autonomy," *The International Journal of Robotics Research*, vol. 17, no. 4, pp. 315–337, 1998.

[7] C. Schlegel and R. Worz, "The software framework smartsoft for implementing sensorimotor systems," in *Intelligent Robots and Systems, 1999. IROS'99. Proceedings. 1999 IEEE/RSJ International Conference on*, vol. 3. IEEE, 1999, pp. 1610–1616.

[8] R. Bischoff, T. Guhl, E. Prassler, W. Nowak, G. Kraetzschmar, H. Bruyninckx, P. Soetens, M. Haegele, A. Pott, P. Breedveld *et al.*, "Brics-best practice in robotics," in *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*. VDE, 2010, pp. 1–8.

[9] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009.

[10] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, vol. 1, 2003, pp. 317–323.

[11] M. Himmelsbach, F. Hundelshausen, and H. Wuensche, "Fast segmentation of 3d point clouds for ground vehicles," in *Intelligent Vehicles Symposium (IV), 2010 IEEE*. IEEE, 2010, pp. 560–565.

[12] B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel, "On the segmentation of 3d lidar point clouds," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2798–2805.

[13] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," *KI-Künstliche Intelligenz*, vol. 24, no. 4, pp. 345–348, 2010.