Particle Filter based SLAM

Aswin P Ajayan

Indian Institute of Technology, Bombay

Jun 30, 2020

Overview

What is SLAM

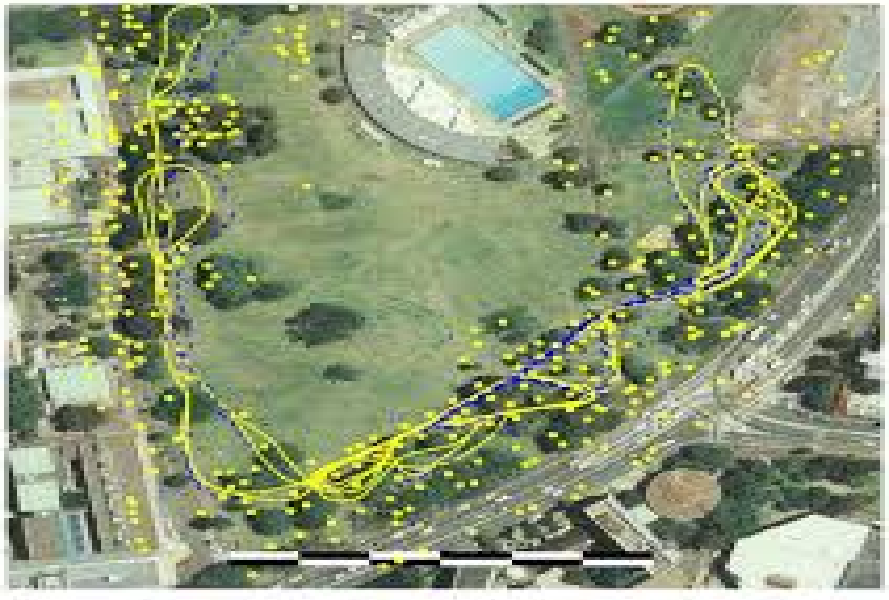
- Computing robot's poses and map environment simultaneously
- Localisation : estimating robots location
- Mapping : building a MAP
- Given
  - $u_{1:T} = \{u_1, u_2, u_3 .... u_T\}$, the control inputs
  - $z_{1:T} = \{z_1, z_2, z_3, ..., z_T\}$, observations
- Wanted
  - m, map of the environment
  - $x_{0:T} = \{x_0, x_1, x_2, ..., x_T\}$, robot location
  - $p(x_{0:T}, m | u_{0:T}, z_{1:T})$, the SLAM posterior

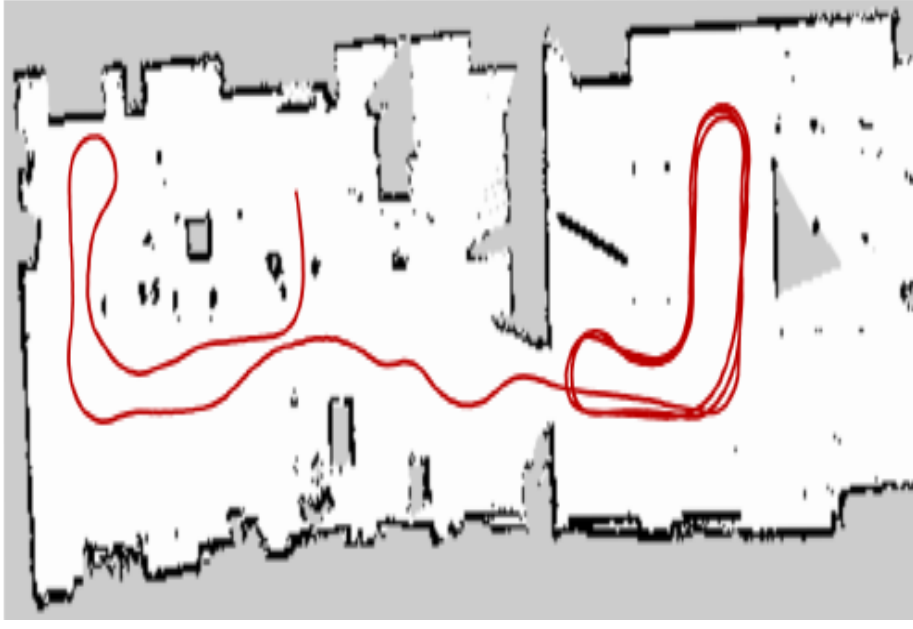
Objective

- To review the literature available on SLAM. To get acquainted with various techniques available for SLAM and examine some of them in depth. Simulation in reallife constraints using ROS.
  Types of SLAM techniques explored includes
- **Kalman Filter based approaches**
  - Extended Kalman Filter
  - Unscented Kalman Filter
  - Extended Information Filter
- **Particle Filter based approaches**
  - Fast SLAM -Rao Blackwellised Particle Filter
    - Augmented MCL
    - Mixture MCL

Types of SLAM

- Full SLAM vs online SLAM
  - Full SLAM $p(x_{0:T}, m|u_{1:t}, z_{1:t})$
  - Full SLAM $p(x_t, m|u_{1:t}, z_{1:t})$
- Feature Based SLAM vs Grid Based





- Active SLAM vs Passive SLAM

Recursive Bayesian Estimation

- **Markov assumption** : next state depends only on the present state

$$p(x_t|x_{0:t}, u_{1:t}) = p(x_t|x_{t-1}, ut)$$

- **Recursive Bayesian Estimation**

$$\begin{aligned} bel(x_t) &= p(x_t|z_{1:t}, u_{1:t}) \\ &= \eta p(z_t|x_t) \int_{x_{t-1}} p(x_t|x_{t-1}, u_t) * bel(x_{t-1}) dx_{t-1} \end{aligned}$$

we can split this into predict and update steps where

Predict Step

$$\overline{bel(x_t)} = \int_{x_{t-1}} p(x_t|x_{t-1}, u_t) * bel(x_{t-1}) dx_{t-1}$$

Update Step

$$bel(x_t) = \eta * p(z_t|x_t) * \overline{bel(x_t)}$$

Bayes filter recursive estimation is realised using kalman filter, Information Filter, particle filter etc.

Kalman Filter Family

Beliefs are represented in parametric form with mean vector μ_t and covariance matrix Σ_t . Closed form expressions are available for belief propagation under certain cases

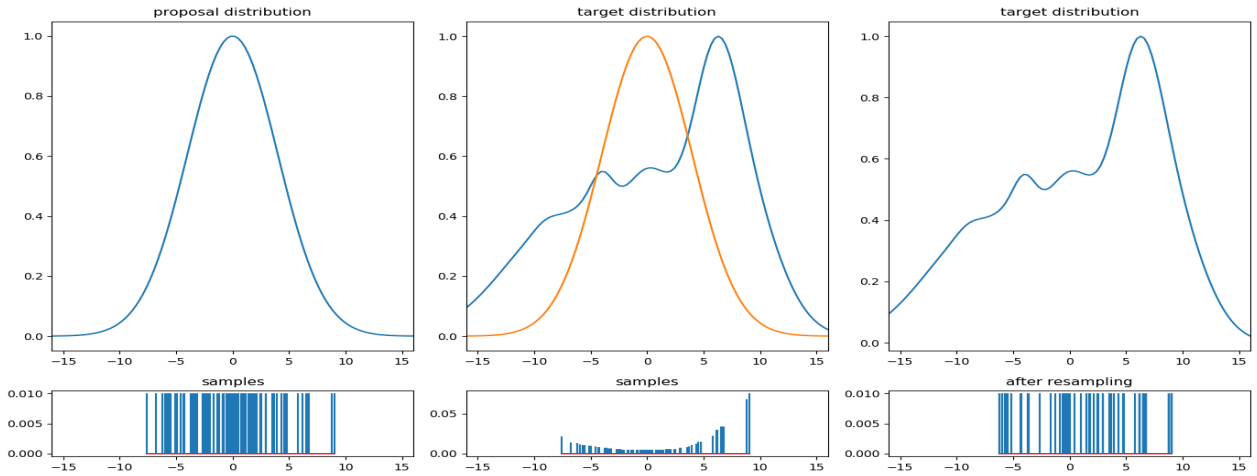
- **Kalman Filter :** Assumes linear models(state transistion and measurement) and gaussian posterior
- **Extended Kalman Filter :** Linearises the model and then applies normal KF
- **Unscented Kalman Filter :** Samples a gaussian posterior from the resulting non linear transformation
- **Information Filters :** Uses canonical representaion of Belief i.e. μ_t^{-1} and Σ_t^{-1}

Particle Filter

- Use particles to represent instead of parametric models.
- Can effectively model mutli modal distributions .
- Can Take care of the non linearities in the model
- Computationally intensive
- Based on importance sampling principal

Particle filter

Importance Sampling - Use to generate samples from an arbitrary distribution



steps

- Draw samples from a proposal distribution
- calculate the importance weight as $w_t^{[j]} = \frac{target(x_t^{[j]})}{proposal(x_t^{[j]})}$
- Resample based on weights

[4] https://github.com/aswinpajayan/seminar-related/scripts/importance sampling.py

Particle filter

- sampling from proposal distribution :

$$x_t^{[j]} \sim \pi(x_t | x_{t-1}, u_{t-1})$$

- Importance weighting :

$$w_t^{[j]} = \frac{target(x_t^{[j]})}{proposal(x_t^{[j]})} = p(z_t | x_t)$$

- Resampling : Draw sample i with probability $w_t^{[j]}$

[1] Source: Probabilistic Robotics, Sebastian Thrun

Rao-Blackwellisation for SLAM - Fast SLAM 1.0

- Particle filters are ideal for low dimensional states i.e. $||x_t||$ is small.
- for SLAM problem, Number of landmarks may be very large. So Particle filter cant be used directly
- Estimate the pose using particle filter and then compute map.

$$
\begin{aligned}
p(a,b) &= p(b|a).p(a) \\
\sim p(x_{0:t}, m_{1:M}|z_{1:t}, u_{1:t}) &= p(x_{0:t}|z_{1:t}, u_{1:t}).p(m_{1:M}|z_{1:t}, u_{1:t}) \\
&= p(x_{0:t}|z_{1:t}, u_{1:t}) \Pi p(m_i|z_{1:t}, u_{1:t})
\end{aligned}
$$

Now each particle respresents a path hypothesis and it has an assosicated map with it. Splitting $p(m_{1:M})$ into $\Pi p(m_i)$ reduces the computational complexity further, each of this is calculated by a 2x2 EKF.

RBPF particle structure

Each particle maintains M 2x2 EKF along with the 3 pose variables

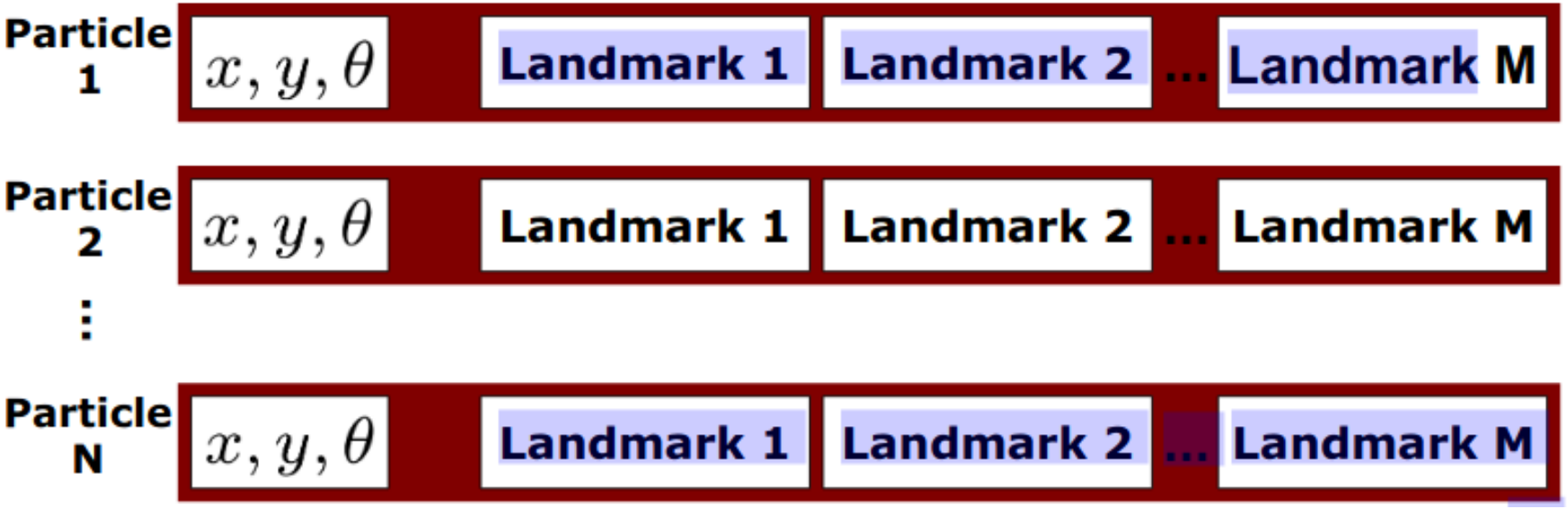
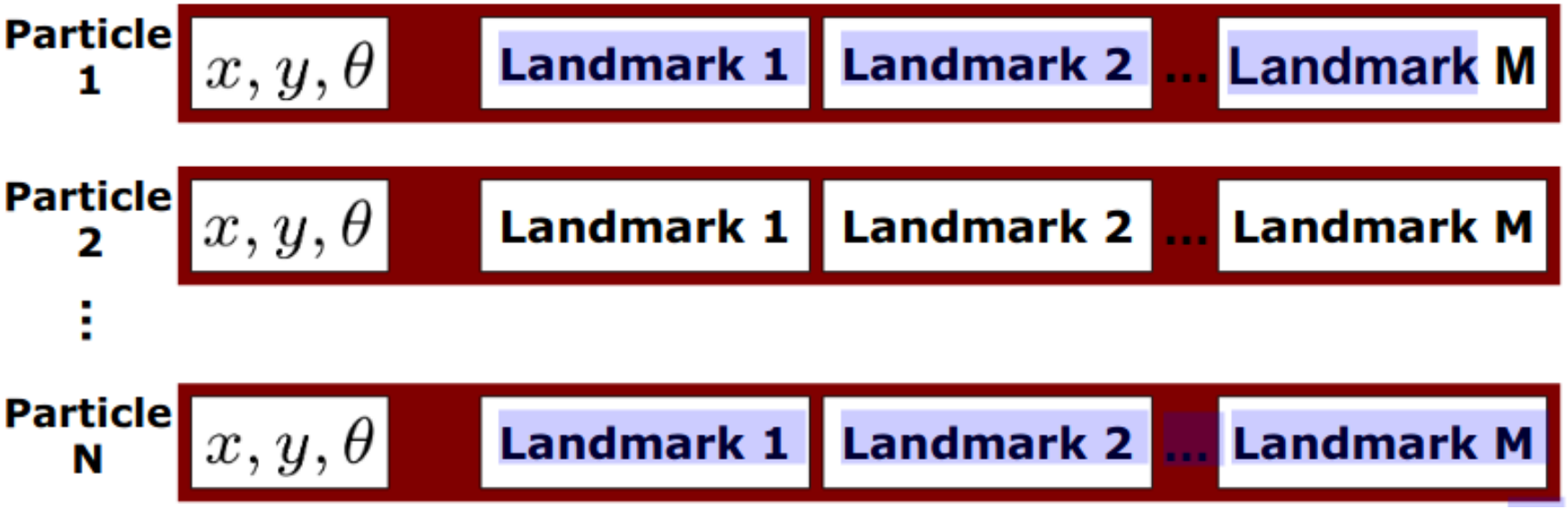


Figure: Particle structure in RBPF-slam

Implementing Particle filter localisation

- **Simulation Platform:** - ROS melodic Ubuntu 18.04
- **Robot:** - Custom robot using URDF
  - motion model libgazebo_ros_diff_drive.so
  - sensor model head_hokuyo_sensor
- **Visualisation:** - matplot lib interactive plots , python 2.7

Creating a custom robot



- *Universal Robot Description Format* provides an easy way to create a custom meshes for a robot. The motion model and sensor model are realised using the gazebo plugins. The robot was created using xacro files: which stands for XML Macros.
- sensor: Laser range finder with standard deviation 0.01 and gaussian model was used

[3] Source: theconstructsim.com

Creating a motion model

- Particle filter uses a motion model as the proposal distribution and sensor model for inferring importance weights
- A simple motion model was used :

$$x_t = x_{t-1} + v.cos(\theta_{t-1})$$

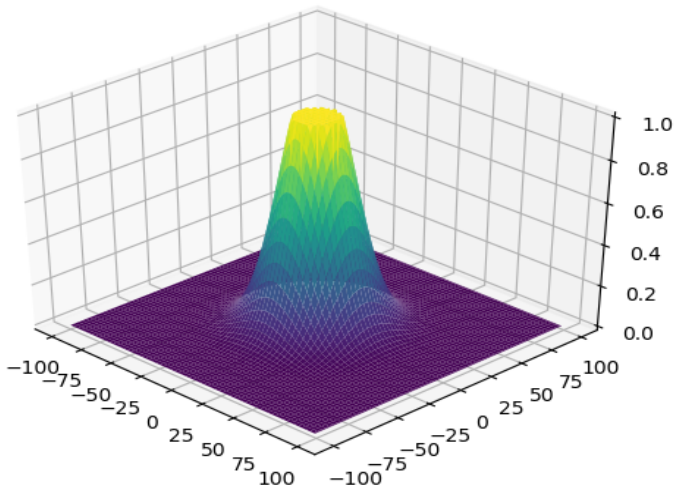
$$y_t = y_{t-1} + v.sin(\theta_{t-1})$$

$$\theta_t = \theta_{t-1} + w\delta_t$$

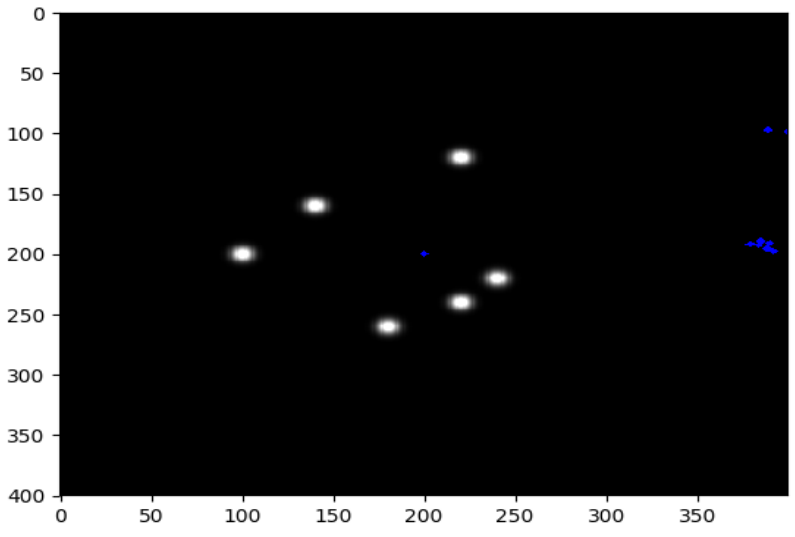
v, w are linear velocity, angular velocity.

Creating a sensor model

- Maximum likelihood field was chosen to be used as the sensor model
- The field was generated using a convolution with a kernel and point landmark map.



field.png

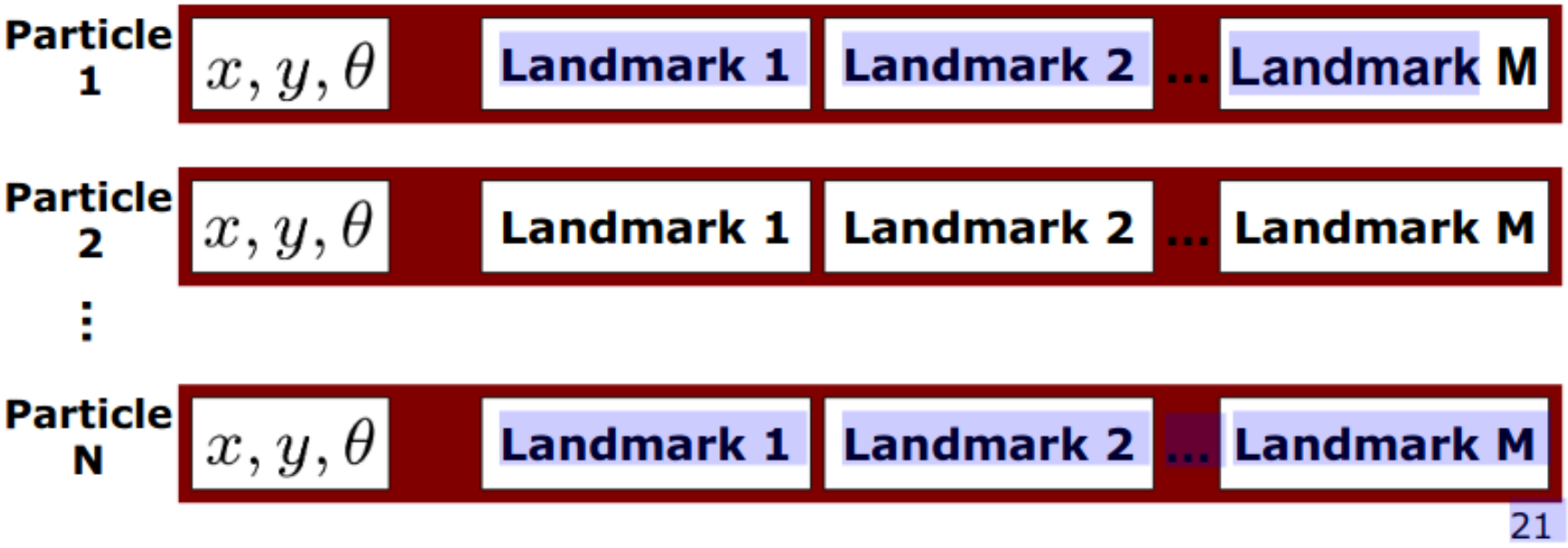


Resampling

Various Resampling approaches were tried out -

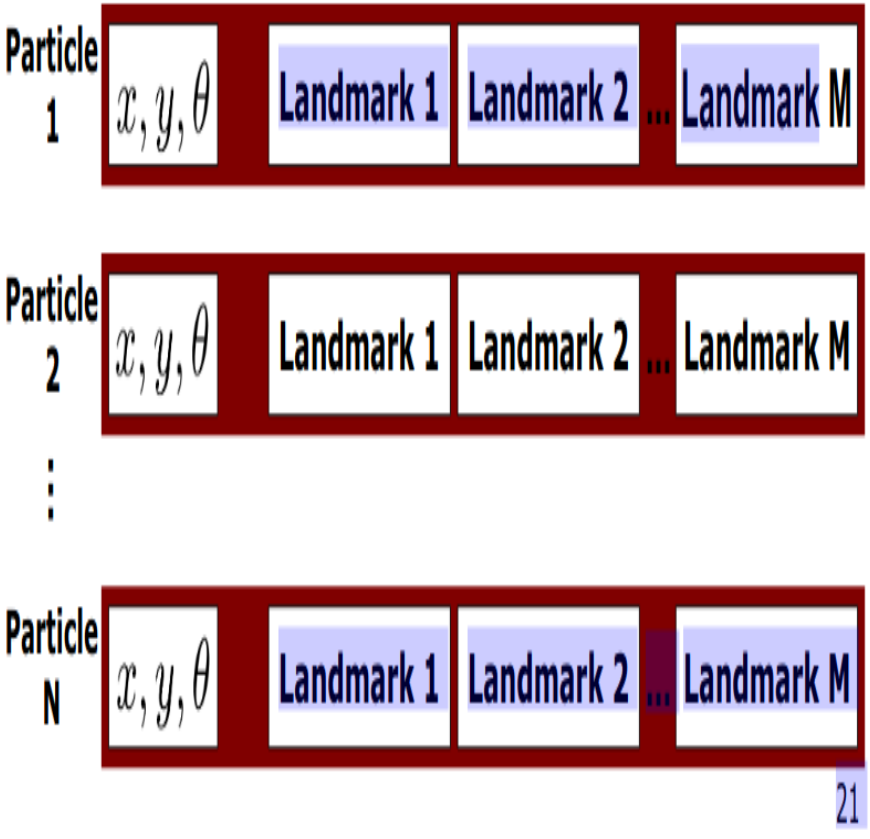
- **Fitness proportion sampling:** Used the default resampler available in numpy
  numpy.random.choice(particles, num=N, p=weights)
- **Roulette wheel selection :** Fitnesss values arranged as CDF around a wheel and a point is chosen at random
- **Stochastic Universal sampling :** Roulette wheel selection will be dominated by a few particles of highest fit. Some particles with high fitness values might be shadowed by the most fit members. This can result is global localisation failure. To avoid this SUS was tried. Though it reduces localisation failure, it can still result in localisation failure

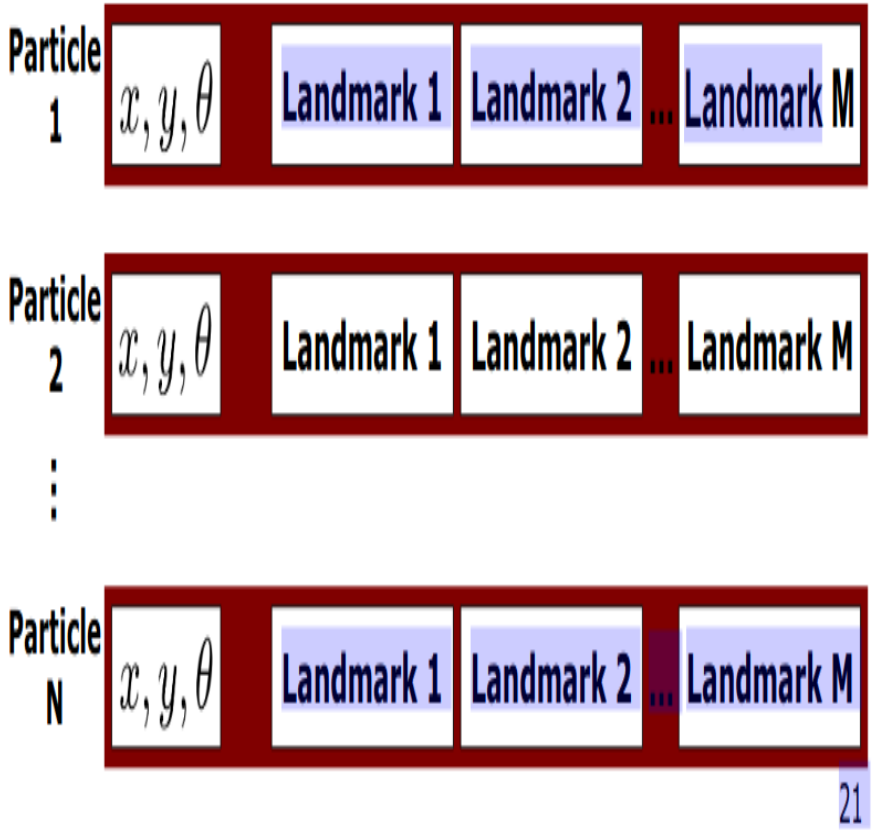
Simulations



21

Analysis of DLC384 Reading FSM Diagram and clock diagram from data sheet

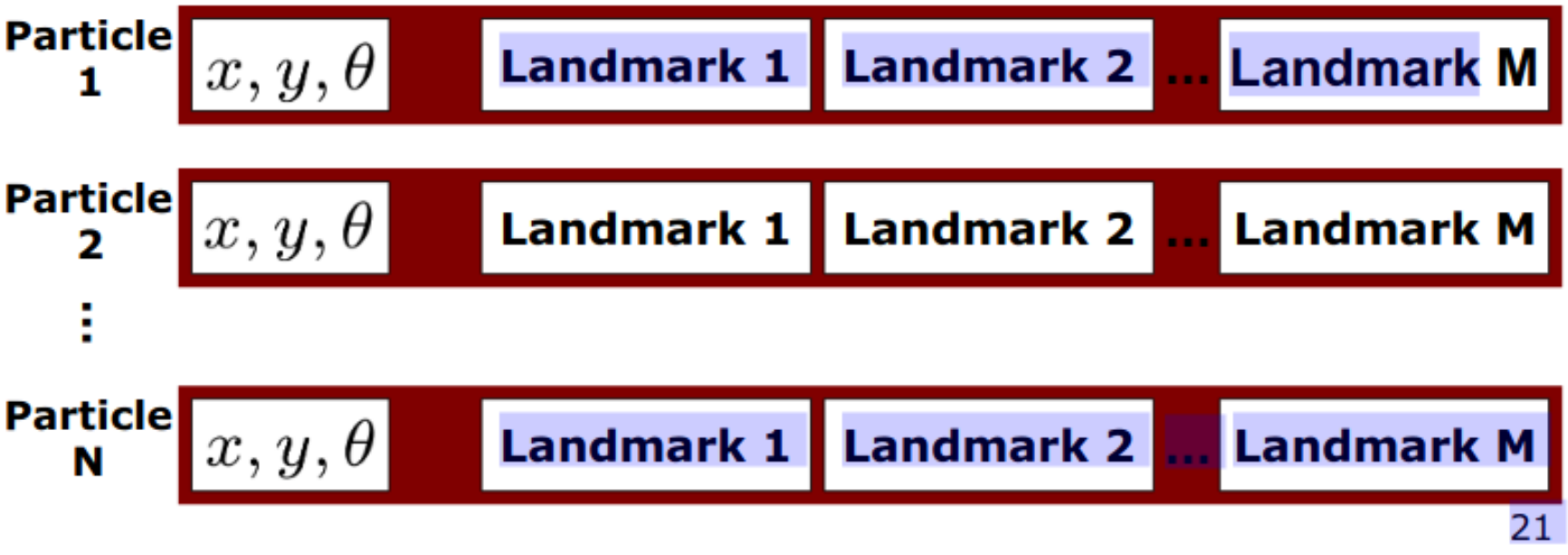




DLC384 Reading FSM

- DLC384 Reading FSM is a finite state machine having 4 states, a) State_Reset; b) State_INT; c) State_EN; d) State_Read.
- States (c) and (d) are two main states, in which FSM toggles maximum amount of time.
- In state (C) FSM waits for VOUT_EN signal from DLC384 which is expected to arrive after 18.5 cycles from INT falling edge.
- In State (d) INT signal is high(for 384 cycles) so that integration of next row pixels can be done. Also since VOUT_EN is high so simultaneously current rows pixels value is captured at every rising edge of clock.

DLC384 Controller FSM Diagram



21

DLC384 Controller

- There are two different input pins used to configure this sensor, SERIAL and SERDAT.
- If SERIAL = 0V then SERDAT is off and CTIA capacitance (on which integration charge accumulates) is fixed to 14 pF.
- If SERIAL = 5 V and SERDAT = 0V then CTIA capacitance is fixed to 18 pF or by sending some bit streams via SERDAT pin we can set different CTIA capacitance and different functions of the sensor can also be changed like(HFLIP,VFLIP).
- As can be seen in FSM diagram of the DLC384 controller, which transmit 51 bits from SERDAT pin if SERIAL pin is high.
- SERIAL pin is connected to DIP pin of FPGA board which can be turn on/off externally.
- Bit streams are hardcoded in FPGA which can be later controlled from DIP switches.

Pin Configuration and it's Key Number

In Port	Pin	Key	Out Port	Pin	Key
resetn	j15	Key0	reset_out	PIN_A2	GPIO_02
inclk	R8	50 MHz osc	int	PIN_A3	GPIO_03
vout_en	PIN_D3	GPIO_00	clock_6.25	PIN_B3	GPIO_04
fr	PIN_C3	GPIO_01	pll_locked	PIN_A15	LED0
Vout_ADC	PIN_A4,B5,A5, D5,B6,A6,B7,D6	GPIO_06 to GPIO_013	serial_out	PIN_B4	GPIO_05
			serdat	PIN_A4	GPIO_06

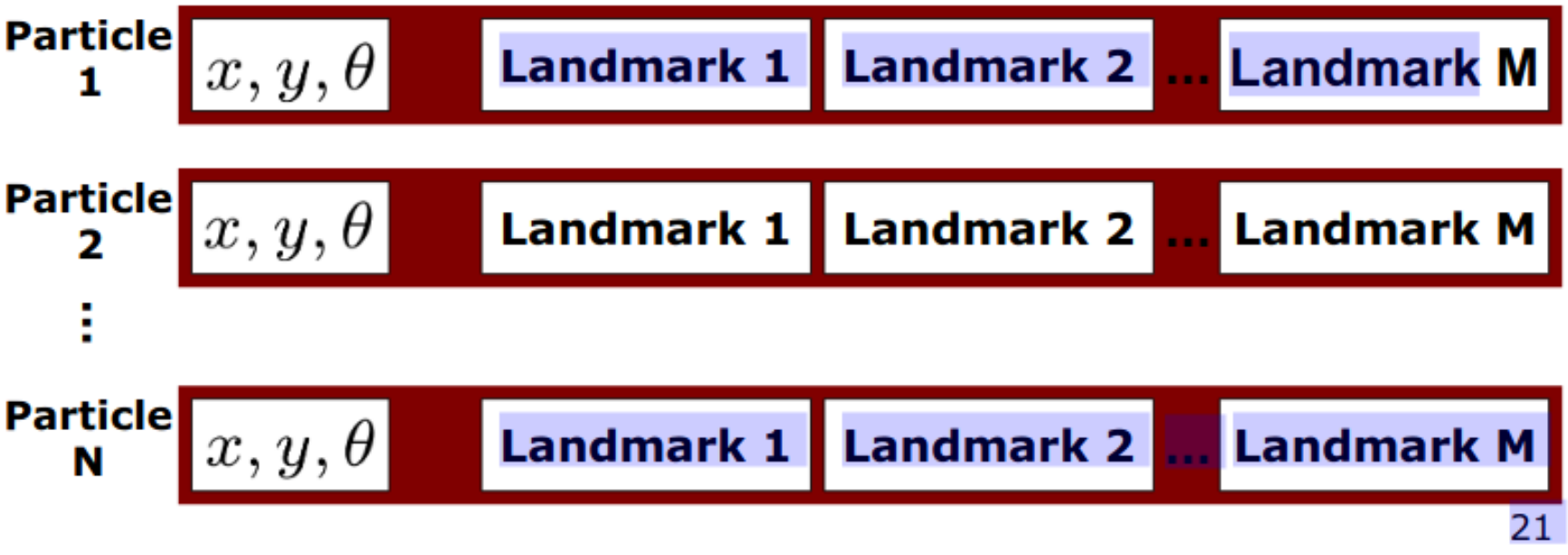
Table: 1

Resources Used

Resource Name	Number/% used
Total Logic Elements	153/22320 (less than 1%)
Total Registers	44
Total Memory Bits	524288/608256 (86%)
Total PLLs	1/4 (25%)

Table: 2

Simulation Results Part-1: Control Signal analysis



Simulation Results Part-1: Control Signal analysis

- To compare the simulated timing pulses of control signals with data sheet of DLC384 it is assumed that in a frame -
  - Number of columns are 5, in place of 384.
  - Number of Rows are 3, in place of 288.
  - Number of cycles after which the next row is read (VOUT_EN is asserted to 1 after INT is asserted to 0) is 2.5 in place of 18.5.
- Simulation timing pulses of control signals (figure-10) and Clock Diagram of DLC384 (figure-6) overlaps.
- we can calculate the pulse duration by looking at following counter values (In Figure-10).
  - For 2.5 cycle(for sensor it is 18.5), for which VOUT_EN is low, look for q_18_cycle.
  - For 5 cycle(for sensor it is 384), for which VOUT_EN and INT are high, look for s_h_counter.
  - For current s_v_count to be seen.

Simulation Results Part-1: Control Signal analysis (continue)

- Using s_v_count and s_h_count values current pixel location/index can be calculated.
- For reset, look for s_reset_out.
- 6.25 MHz clock is s_clock_6_out.
- Signal fr marks the start of a new frame, after reset is done.
- If "s_serial" is high then "s_serial_out" is set to high and at "s_serdat" configuration bits are sent.

Simulation Results Part-2: Working on actual images stored in a text file

- 3 images of same pixel size is taken, considered as 3 consecutive frames of a video.(img_1.png, img_2.png, img_3.png)
- Using a python script, images are converted into a single vector and then saved in a file one after another. This file will simulate the camera pixels.(Python script - image_to_text.ipynb , file - img_1_2_3.txt)
- In Quartus, a test bench is created which simulates the Camera sensor DLC384 sensor and read file img_1_2_.txt for throwing pixels value.
- Sensor DLC384 gives analog value at VOUT pin, so a ADC considered b/w the sensor and the camera driver.
- Once a frame is sent, testbench reads the Dual Port RAM and saves the data in another file output_img.txt.
- After the simulation both img_1_2_3.txt and out_img.txt files are compared using a python script, which shows both matches perfectly.

Simulation Results Part-3: DLC384 Controller

- DLC384 Controller simulation can be seen in figure-10, where s_serial_out and s_serdat should be connected to SERIAL and SERDAT pins of the DLC384 sensor.

Future Work

- Built an interface between Dual Port RAM(ping_pong concept can also be introduced) and VGA port, so the video can be observed on a VGA ,monitor.
- DeO-Nano development board give 3.3 V as high and 0V as low, but Sensor pins works at 5V(as high) and 0V(as low) so level-shifter circuit need to be used.

References

[1] https://www.opli.net/opli_magazine/imaging/2017/sierra-olympic-introduces-world-first-true-hd-thermal-camera-jan-news/

[2] https://in.element14.com/terasic-technologies/p0082/e22f17c6n-DEO-nano-dev-kit/dp/2076463

[3] http://www.123-cctv.com/CCTV-Monitors/7-inch-Security-LCD-Monitor-with-VGA.html

[4] DLC384 User Manual

[5] A Design of Readout Circuit for 384x288 Uncooled Microbolometer Infrared Focal Plane Array - 2012 IEEE 11th International Conference on Solid-State and Integrated Circuit

[6] Large dynamic range Readout Integrated Circuit for Infrared Detectors -2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)

Thank You for your Attention