بسم الله الرحمن الرحیم

موضوع پروژه

کاربرد جبر خطی در نرم افزار اندرویدی که دمای N نقطه را با استفاده از دیتا فیوژن به دست آورد

استاد

دکتر میگلی

حل تمرین

غلامرضا قاسمی

تهیه کننده

علی قنبری ۹۷۰۲۱۶۶۵۷

فهرست

مقدمه **3**

تلفیق داده چیست؟ 3
ظرورت استفاده از تلفیق داده 3
منشأ نظریه تلفیق داده 3
فیلتر کالمَن 3

شبیه سازی **4**

جسم مجازی 4
انتقال داده به سنسور ها 5
نحوه تولید ماتریس دمای سطح جسم 5
سنسور مجازی دو بعدی 6
ایجاد نویز در سنسور مجازی 7
معادله تشابه ماتریس ها 8

فیلتر کالمَن **9**

پیش بینی 10
بروزرسانی 10
سنسور ادغام 11

برنامه اندروید **12**

قابلیت ها 12
صفحه ها 13

مرجع ها **15**

مقدمه

هدف این پروژه استفاده از جبر خطی برای تلفیق داده است. یکی از روش های مهم تلفیق داده به نام فیلتر کالمن که بر اساس جبرخطی است در این پروژه استفاده شده. در این گزارش مفاهیم استفاده شده و نحوه پیاده سازی آن ها توضیح داده شده اند.

تلفیق داده چیست؟

دیتا فیوژن یا تلفیق داده یک فرایندی است که در آن داده های حاصل از چندین منبع ادغام می شوند تا اطلاعات دقیق تر و یکسان تری نسبت به داده های حاصل از هر یک از منبع ها به صورت جداگانه حاصل شود.

ظرورت استفاده از تلفیق داده

چون ما چندین ورودی داریم در صورتی که یکی از ورودی ها خراب شود یا در دسترس نباشد، سیستم می تواند پایداری خود را حفظ کند، همچنین ادغام چندین منبع داده باعث کاهش چشم گیری در نرخ داده های نا مطمئن و افزایش اطمینان می شود.

منشأ نظریه تلفیق داده

مفهوم تلفیق داده منشأ گرفته از قابلیت انسان ها و حیوانات برای استفاده ترکیبی از حواس خود هستند. مثلا میتوانند با ترکیب اطلاعاتی که از بینایی، لمس و بویایی از یک ماده دریافت می کنند، تصمیم بگیرند که این ماده خوراکی است یا نه.

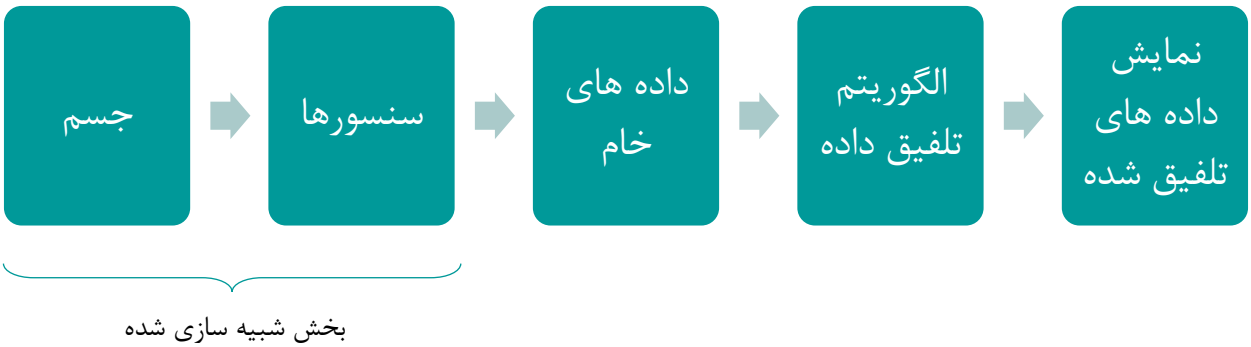
فیلتر کالمَن (Kalman filter)

این فیلتر یکی از الگوریتم هایی است که برای تلفیق داده استفاده می شود و می تواند بر اساس حالت فعلی، مدل و داده های جدید، یک پیش بینی دقیق تری از حالت فعلی جسم به ما بدهد.

شبیه سازی

برای اینکه بتوانیم از فیلتر کالمن استفاده کنیم، باید به آن داده خام بدهیم. بجای اینکه یک سنسور واقعی را به برنامه خود وصل کنیم می توانیم رفتار یک سنسور واقعی را شبیه سازی کنیم.

در نهایت ساختار برنامه اینگونه خواهد بود:



در برنامه ساخته شده فقط بخش های سنسور و جسم، شبیه سازی شده هستند و می توانیم با جایگزین کردن این بخش ها با معادل واقعی آن ها، همچنان از برنامه ساخته شده استفاده کنیم.

در بخش های بعدی، نحوه پیاده سازی مربوط به آن بخش آمده است:

جسم مجازی

سنسورهای مجازی به یک جسم مجازی نیاز دارند که باید در دسترس آن ها قرار بگیرد تا بتوانند آن را اندازه گیری کنند. برای همین از جسم مجازی شروع می کنیم. برای اینکار یک کلاس به نام VirtualObject2D در برنامه تعریف کرده ام که نماینده جسم مجازی ما خواهد بود و شکل ساده شده پیاده سازی آن بصورت زیر است:

```
class VirtualObject2D {
  final double minTemp;
  final double maxTemp;
  final Matrix surfaceTemps;
...
}
```

VirtualObject2D سه مؤلفه دارد:

minTemp و maxTemp که کمترین و بیشترین عددی که ممکن است در ماتریس دما باشد را نشان می دهند.

surfaceTemps که یک ماتریس $n \times m$ است و دمای همه‌ی نقاط جسم را ذخیره می کند.

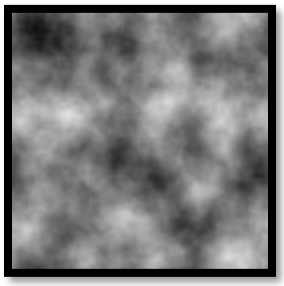
انتقال داده به سنسور ها

برای انتقال داده به سنسور ها، به جای اینکه دمای جسم توسط سنسورها خوانده شود، ماتریس دما از جسم به سنسور ها فرستاده می شود. اینکار باعث می شود که بتوانیم چیزی را که سنسور ها می توانند ببینند از طرف جسم مجازی قابل کنترل باشد.

اگر بخواهم بصورت فنی تری این فرایند را توضیح بدم، جسم مجازی یک Stream<Matrix> یا پخش جریانی ماتریس دارد و سنسور ها می توانند به این پخش جریانی گوش دهند و موقعی که جسم مجازی، به این جریان ماتریسی اضافه کند، به گوش همه سنسورها می رسد و آن ها می توانند خروجی خود را بر اساس آخرین ماتریسی که از راه جریان به آن ها رسیده آپدیت کنند.

نحوه تولید ماتریس دمای سطح جسم

برای تولید ماتریس دمای سطح یک جسم مجازی از الگوریتم پرلین نویز (Perlin noise) استفاده کردم چون خروجی این الگوریتم بصورت تپه ای است و در یک بازه کوچک نقطه از 0 به 1 نمی رود. این الگوریتم در n بعد کار می کند و برای تولید مرحله و محیط های شبه طبیعی در بازی سازی خیلی کاربرد دارد. عکس زیر یک نمونه پرلین نویز دو بعدی است که در آن هر نقطه مقداری بین 0 و 1 دارد:



چون ما مینیمم و ماکسیمم دمای جسم را داریم می توانیم دمای هر نقطه را با استفاده از فرمول زیر روی یک پرلین نویز دو بعدی بدست آوریم:

$$SurfaceTemps_{ij} = (t_{ij} \times (maxTemp - minTemp) + minTemp)$$

برای مثال اگر یک پرلین نویز 3×3 داشته باشیم و کمترین و بیشترین دما 273 و 400 باشند، یک نمونه خروجی که می توانیم داشته باشیم به این صورت است:

$$\begin{bmatrix} 287 & 281 & 273 \\ 279 & 273 & 281 \\ 273 & 279 & 287 \end{bmatrix}$$

سنسور مجازی دو بعدی

این سنسور قرار رفتار یک دوربین حرارتی را شبیه سازی کند و بصورت $n \times m$ سنسور یک بعدی طراحی شده. کلاسی که نماینده این نوع سنسور خواهد بود VirtualTempSensor2D نام دارد و شکل ساده شده پیاده سازی آن بصورت زیر است:

```
class VirtualTempSensor2D {
  final double errorRate;
  final VirtualObject2D object;
  bool enabled;
  double accuracy;
  double avgAccuracy;
  bool kalmanFilter;
  int updates;
  double totalAccuracy;
  Stream<Matrix> temps;
}
```

VirtualTempSensor2D از نظر پیاده سازی، پیچیده ترین بخش برنامه است و این موضوع از تعداد مؤلفه های آن پیداست که کاربر هر یک از آنها به این صورت است:

errorRate دلتای خطای سنسور را مشخص می کند که در بخش نحوه تولید نویز سنسور بیشتر در مورد آن خواهیم پرداخت.

object به ما دسترسی مستقیم به جسم مجازی می دهد تا بتوانیم به تغییرات آن گوش دهیم و ماتریسی که بعد از اعمال نویز بدست آوردیم را با دمای واقعی جسم مقایسه کنیم تا بتوانیم دقت سنسور را محاسبه کنیم.

enabled نشان می دهد که سنسور روشن است یا خاموش که در برنامه فعلی سنسور ها همیشه روشن هستند.

accuracy یک مقدار متغییر است و در هر باری که سنسور آپدیت می شود، تغییر می کند. این متغییر دقت سنسور در آخرین بروزرسانی را نمایش می دهد.

avgAccuracy همانند accuracy تغییر می کند ولی با این تفاوت که متوسط دقت سنسور، از زمان ساختش را نمایش می دهد و بصورت زیر محاسبه می شود:

$$avgAccuracy = \frac{totalAccuracy}{updates}$$

kalmanFilter نشان می دهد که فیلتر کالمن روشن است یا خاموش. با تغییر این پارامتر می توانیم، فیلتر کالمن سنسور را خاموش یا روشن کنیم.

updates شمارنده تعداد آپدیت های سنسور است که برای محاسبه دقت متوسط استفاده می شود.

totalAccuracy مجموع همه‌ی دقت های سنسور از شروع تا آخرین بروزرسانی است.

temps که جریان خروجی سنسور است و برای ورودی سنسور ادغام و رابط کاربری استفاده می شود.

ایجاد نویز در سنسور مجازی

دوربین حرارتی مجازی ما قرار است که یک ماتریس $n \times m$ بگیرد و مشابه یک دوربین حرارتی واقعی، آن را با مقداری خطا به ما بازگرداند.

دوربین های حرارتی واقعی به دو صورت نویز دارند:

۱. دوربین کالیبره نباشد و دمای همه نقاط بصورت $t_n \pm \Delta$ گزارش می شوند.
۲. بدلیل خواص جسم یا گرد و غبار موجود در هوا چندین نقطه با خطای $\pm\Delta$ گزارش می شوند.

برای شبیه سازی مورد اول، کافی است که همه درایه های ماتریسی که نماینده دمای جسم است را با مقدار $\pm\Delta$ جمع کنیم کنیم. مثلا اگر ماتریس دمای سطح جسم به این شکل باشد:

$$\begin{bmatrix} 302.0 & 302.4 \\ 302.6 & 303.0 \end{bmatrix}$$

آنگاه یک Δ برای آن فریم در نظر می گیریم که متناسب با $\pm errorRate$ سنسور باشد، مثلا اگر میزان خطا 30 باشد، Δ که در نظر می گیریم می تواند عددی بین 30- و 30 باشد و خروجی مرحله اول بصورت نمونه، بشکل زیر خواهد بود:

$$\begin{bmatrix} 302.0 & 302.4 \\ 302.6 & 303.0 \end{bmatrix} + \begin{bmatrix} 1.5 & 1.5 \\ 1.5 & 1.5 \end{bmatrix} = \begin{bmatrix} 304.5 & 303.9 \\ 304.1 & 304.5 \end{bmatrix}$$

در مرحله دوم چندین نقطه بصورت تصادفی از خروجی مرحله اول انتخاب می شوند و با $\pm\Delta$ جمع می شوند که در اینجا Δ بصورت تصادفی برای هر نقطه تولید می شود و ممکن است اثرات خطا در مرحله قبل را خنثی کند. مثلا به یکی از نقاط 2- اضافه می کنیم:

$$\begin{bmatrix} 304.5 & 303.9 \\ 304.1 & 304.5 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ -2 & 0 \end{bmatrix} = \begin{bmatrix} 304.5 & 303.9 \\ 302.1 & 304.5 \end{bmatrix}$$

در اینجا می بینیم که دما در نقطه‌ی t_{21} به مقدار واقعی آن نزدیکتر شد.

معادله تشابه ماتریس ها

برای اینکه بدانیم که تلفیق داده، دقت سنسور های ما را بالا می برد نیاز داریم که دقت سنسور ها را اندازه گیری کنیم. برای بدست آورن درصد تشابه دو ماتریس، یک تابع مشابهت (similarity function) بصورت زیر نوشته شده:

فرض کنیم دو ماتریس هم اندازه با نام ماتریس اصلی S و ماتریس نویز دار N باشد و min و max نشان دهنده کمترین و بیشترین دمای ممکن برای جسم باشند، معادله زیر میزان تشابه ماتریس نویز دار به ماتریس اصلی را حساب می کند:

$$\text{accuracy} = \frac{\sum_i^{rows}\sum_j^{columns}\left(1 - \left|\frac{S_{ij} - min}{\max - min} - \frac{N_{ij} - min}{\max - min}\right|\right)}{rows \times columns}$$

تابع مشابهت نوشته شده از نوع باینری است و accuracy عددی بین 0 و 1 خواهد بود.

فیلتر کالمَن (Kalman filter)

این فیلتر یکی از الگوریتم هایی است که برای تلفیق داده استفاده می شود و می تواند بر اساس حالت فعلی، مدل و داده های جدید، یک پیش بینی دقیق تری از حالت فعلی جسم به ما بدهد.

برای اینکه از این الگوریتم برای سنسورهایمان استفاده کنیم ابتدا باید مدل سیستم را طراحی کنیم. مدلی که طراحی کردم به این صورت است:

$$t_k = t_{k-1} + \dot{t}_{k-1}\Delta T + \frac{Q}{mc}(\Delta T)^2 + V_1$$

$$\dot{t}_k = \dot{t}_{k-1} + \frac{Q}{mc}(\Delta T) + V_2$$

$$V = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}$$

در این جا، Q انرژی وارد به سیستم بر حسب ژول، m جرم جسم، c ظرفیت گرمایی ویژه جسم می باشد. V بردار خطا ناشی از خطاهای خارجی و ΔT فاصله زمانی از بروزرسانی قبلی بر حسب ثانیه است.

چون ورودی الگوریتم بصورت ماتریسی است، باید فرمول های بالا را بصورت ماتریسی بنویسیم:

$$r_k = \begin{bmatrix} t_k \\ \dot{t}_k \end{bmatrix} = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} t_{k-1} \\ \dot{t}_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{(\Delta T)^2}{mc} \\ \frac{\Delta T}{mc} \end{bmatrix} Q + V_k$$

معادله بالا فرم کلی $r_k = Fr_{k-1} + Bu_k + V_k$ را دارد که ماتریس F در مرحله پیش بینی برای ما خیلی مهم است:

$$\rightarrow F = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix}$$

این فیلتر از دو ماتریس r و P تشکیل شده که به ترتیب نماینده حالت فعلی و کواریانس حالت پیش بینی شده است.

این الگوریتم در دو قسمت کار می کند، پیش بینی و بروزرسانی که در ادامه فرمول‌های کار هر یکی از این مرحله ها آمده است.

پیش بینی

در این مرحله تغییرات مربوط به زمان با استفاده از ماتریس F، بصورت زیر، حالت سیستم پیش بینی می شود:

$$r_k = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} t_{k-1} \\ \dot{t}_{k-1} \end{bmatrix}$$

همچنین ماتریس واریانس داده ها را هم بروز می کنیم:

$$P = FPF^t + GVG^t$$

ماتریس V مربوط به اغتشاشات(Disturbance) است.

ماتریس های V و G را ماتریس یکه 2x2 در نظر گرفتم.

بروزرسانی

در این مرحله که به آن مرحله اندازه گیری هم گفته می شود می توانیم داده های جدید را وارد کنیم.

معادله بروزرسانی حالت به این صورت تعریف می شود:

$$z_k = t_k + R_k \rightarrow z_k = Hr_k + R_k$$

ورودی های معادله بالا را بصورت زیر محاسبه کردم:

H که مدل اندازی گیری است، چون از اندازه گیری داده ای نداریم، آن را برابر با ماتریس یکه قرار دادم:

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

R که نشان دهنده کواریانس اندازه گیری است (مربوط به نویز) را برابر با 100 برابر ماتریس یکه قرار دادم و طبق اعدادی که امتحان کردم، مقدار دقیق آن زیاد مهم نیست و فقط کافی است که عدد نسبتاً بزرگی انتخاب شود و الگوریتم کالمن باز هم می تواند نتایج دقیقی به ما بدهد:

$$R = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$$

r_k هم داده جدید دریافتی از سنسور است و چون در سیستم فعلی تغییر دما نداریم $\dot{t}_k$ را صفر می گذاریم و برای t_k، دمای دریافتی از سنسور یعنی t_s را قرار می دهیم:

$$r_k = \begin{bmatrix} t_s \\ 0 \end{bmatrix}$$

بعد از جمع کردن همه ورودی های مورد نیاز، باید حالت فعلی را محاسبه کنیم.

ابتدا S که کواریانس اندازه گیری ها است را محاسبه می کنیم:

$$S = HPH^t + R$$

بعد با استفاده از S، نتیجه بهینه کالمن را محاسبه کنیم:

$$K = PH^tS^{-1}$$

و در آخر ماتریس حالت و کواریانس را بروزرسانی می کنیم:

$$P_k = (I - KH)P_{k-1}$$

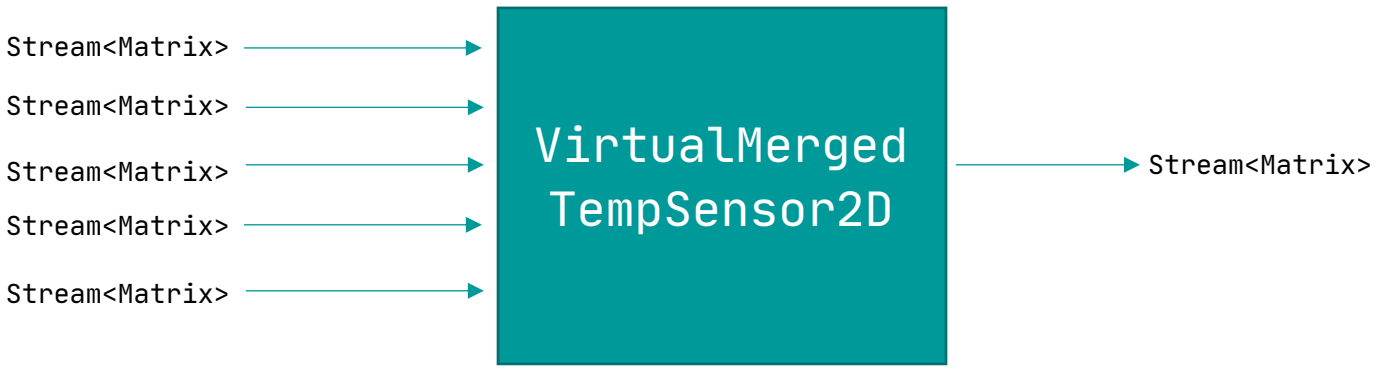
$$r_k = r_{k-1}(K(Z - (H \times r_{k-1})))$$

سنسور ادغام

با استفاده از فیلتر کالمن و یک سنسور می توانیم نتایج خیلی دقیقی بدست بیاوریم. حالا می خواهیم داده های دریافتی از چندین سنسور را با هم ادغام کنیم. برای این کار، یک نوع سنسور مجازی دیگر معرفی می کنم.

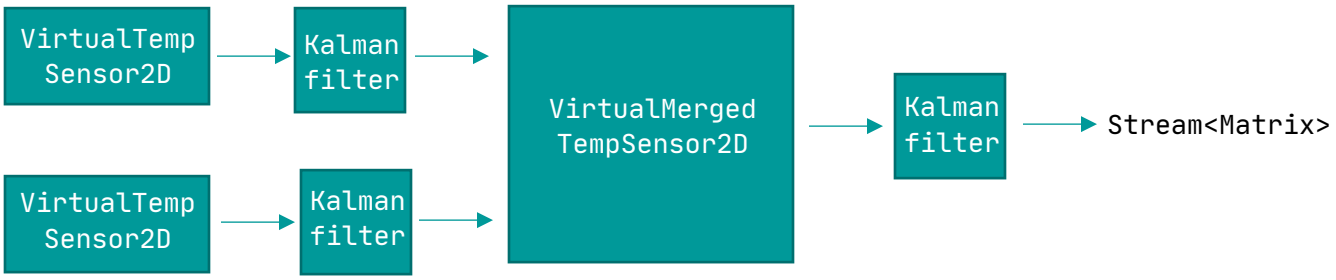
VirtualMergedTempSensor۲D

این سنسور بجای اینکه از یک جسم ورودی بگیرد، از چندین سنسور ورودی می گیرد.



علاوه براینکه داده های خیلی زیادی وارد این سنسور می شوند، می توانیم با اعمال فیلتر کالمن دقت داده ها را بصورت قابل توجهی بالا ببریم.

برای استفاده از فیلتر کالمن و برای بدست آوردن دقیق ترین نتیجه، داده ها را در دو جا فیلتر می کنیم، یک بار قبل از ورود و یک بار بعد از خروج داده از سنسور ادغام.



برنامه اندروید

با استفاده از این برنامه شما می توانید سنسور مجازی بسازید و تفاوت دقت سنسور را در حالت روشن و خاموش بودن فیلتر کالمن، بطور زنده مشاهده کنید.

برنامه اندرویدی این پروژه Linear Algebra Kalman یا بطور خلاصه LAK نام دارد. این برنامه با استفاده از فریمورک فلاتر (Flutter) برای اندروید ساخته شده.

قابلیت ها

۱. قابلیت متوقف کردن و ریست کردن شبیه سازی
۲. قابلیت دیدن دمای واقعی جسم مجازی
۳. قابلیت تولید مجدد ماتریس دمای سطح جسم
۴. قابلیت تعیین کمترین و بیشترین دمای ممکن برای سطح جسم
۵. قابلیت تنظیم نرخ ارسال داده از جسم مجازی به سنسورها
۶. قابلیت تعیین ابعاد ماتریس مبدأ
۷. قابلیت اضافه و کم کردن سنسور مجازی
۸. قابلیت دیدن خروجی سنسور مجازی

۹. قابلیت روشن/خاموش کردن فیلتر کالمن

۱۰.قابلیت تعیین مقدار نویز سنسور

۱۱.نمایش دقت فعلی سنسور

۱۲. نمایش متوسط دقت سنسور

۱۳. نمایش گراف دقت سنسور در ۵۰ بروزرسانی اخیر

۱۴.قابلیت تعیین کواریانس فیلتر کالمن

۱۵.قابلیت اعمال فیلتر کالمن دوم با استفاده از سنسور ادغام

صفحه ها

در این قسمت مهمترین صفحه های برنامه را می توانید:

صفحه اصلی

صفحه جسم مجازی

دکمه ریست

این دکمه شبیه سازی را ریست می کند که جسم جدید با پارامترهای جدید تولید می کند و جایگزین جسم قبلی می کند. با زدن این دکمه، فقط مقداری از شبیه سازی ریست می شود.

ابعاد جسم

تغییر دادن این پارامترها، تأثیری روی شبیه سازی فعلی ندارد و تغییرات، در شبیه سازی بعدی اعمال می شوند.

نمایش دهنده دما

این قسمت از رابط کاربری محتوایات ماتریس دمای سطح جسم را نمایش می دهد.

نرخ ارسال داده ها

این مقدار تعیین می کند که جسم مجازی باید چقدر تا ارسال داده بعدی منتظر بماند.

تنظیمات دما

پارامترهای کمترین و بیشترین دما که برای تولید جسم جدید استفاده می شوند، از طریق این دو ورودی قابل تغییرند.

صفحه سنسور مجازی

خروجی سنسور

این قسمت، خروجی سنسور را پس از اعمال نویز و فیلتر کالمن(در صورت روشن بودن) بصورت زنده نمایش می دهد.

میزان نویز

این ردیف $\pm\Delta$ سنسور را نمایش می دهد.

دقت سنسور

در این دو ردیف دقت فعلی و متوسط دقت سنسور بطور زنده نمایش داده می شوند.

حذف سنسور

با زدن روی این دکمه، سنسور حذف می شود.

فیلتر کالمن

با تغییر وضعیت این سوئیچ می توانید فیلتر کالمن را برای این سنسور روشن و خاموش کنید.

نمودار دقت

این نمودار دقت فعلی سنسور را بعنوان عددی بین ۰ و ۱ بصورت زنده نمایش می دهد.

صفحه سنسور ادغام

این صفحه مشابه صفحه سنسور مجازی دوبعدی است با این تفاوت که ورودی از سنسور های ساخته شده است و هر چه سنسورهای بیشتری داشته باشیم، این سنسور بیشتر آپدیت می شود.

مرجع ها

https://www.slideshare.net/faradars/data-fusion-۶۱۴۰۶۱۶۹

https://en.wikipedia.org/wiki/Data_fusion

https://blog.faradars.org/similarity-and-distance-matrix

https://www.youtube.com/watch?v=xv۱۷xrr۱۱CA